EQUIPO T1

TALLER REFACTORING

INTEGRANTES Y ASIGNACIONES:

-Miguel Angel Licea Cespedes (líder)

-Yiam Oswaldo Rodriguez Delgado (integrante 1)

-Cristopher Leonardo Jaramillo Cumbicos (Integrante 2)

-Luis Alfredo Rodriguez Chavez (integrante 3)

-Carlos Humberto Meneses Murillo (integrante 4)

Enlace repositorio github:

https://github.com/miglcesp01/TallerCodeSmells.git

Contents

Code Smell: Long Parameter List

Código antes de aplicar la técnica de Refactorización:

```
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<>();
}
```

Consecuencias:

- Es difícil entender estos métodos con grandes listas de parámetros.
- Se puede convertir en métodos contradictorios y difíciles a medida de que se alargan.
- Se pierde relaciones entre objetos debido a la disminución de dependencias.

Solución usando la técnica de Refactorización: Introduce Parameter Object

Podemos crear una clase Persona en la que nos ayudará en crear el Profesor, y al constructor del profesor ya no tenemos que pasarle tantos parametos, sino solo un objeto tipo Persona.

```
public class Persona {
    private String nombre;
    private String apellido;
    private int edad;
    private String direccion;
    private String telefono;

    public Persona(String nombre, String apellido, int edad, String direccion, String telefono) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.edad = edad;
        this.direccion = direccion;
        this.telefono = telefono;
    }

    public String getNombre() {...3 lines }
    public void setNombre(String nombre) {...3 lines }
    public String getApellido() {...3 lines }
    public void setApellido(String apellido) {...3 lines }
    public int getEdad() {...3 lines }
    public void setEdad(int edad) {...3 lines }
    public String getDireccion() {...3 lines }
    public void setDireccion(String direccion) {...3 lines }
    public String getTelefono() {...3 lines }
    public void setTelefono(String telefono) {...3 lines }


}
```

Contructor de la clase Profesor luego de la solución:

```
public class Profesor{
    private String codigo;
    private Persona persona;
    private InformacionAdicionalProfesor info;
    private ArrayList<Paralelo> paralelos;

    public Profesor(String codigo, Persona persona) {
        this.codigo = codigo;
        this.persona = persona;
        paralelos= new ArrayList<>();
    }

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

     public double calcularSueldo(){
        double sueldo=0;
        sueldo = info.getAñosdeTrabajo()*600 + info.getBonoFijo();
        return sueldo;
    }
}
```

Code Smells: Data class y inappropriate intimacy

Código antes de aplicar la técnica de Refactorización:

```
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;
    }
}
```

Data class se encuentra en InformacionAdicionalProfesor ya que es una clase con algunos atributos públicos, pero no se encarga de ninguna interacción o de tener algún tipo de comportamiento. Mientras que en el método de calcularSueldo de la clase calcularSueldoProfesor este solo a hace uso de los datos de la otra clase y no utiliza ninguno suyo.

Consecuencias:

- Al tener los atributos públicos cualquiera podría modificar y obtener la información sin tener que haber una comprobación previa.
- El mantenimiento de clase se vuelve mas complicado.
- No se va a tener una buena organización de código con la finalidad de poder reutilizar código.

Solución usando la técnica de Refactorización: Encapsulate Field, Inline class, Inline Temp

Usamos la táctica de refactorización Encapsulate Field para poder tener un mejor encapsulamiento de la clase y que no se puedan acceder de formar inmediata a los atributos. También lo mejor seria aplicar la técnica inline class a la clase de calcularSueldoProfesor con (Con esto se libera memoria operativa de la computadora y clases no necesarias) ya que el único método que este tiene el de calcular sueldo utiliza mas las variables de la clase de info, esto con la finalidad de poder añadir comportamiento a la data class. De paso, al método calcular sueldo se le aplica el inline temp para eliminar las variables temporales.

Código luego de la refactorización:

```java
public class InformacionAdicionalProfesor {
    private int añosdeTrabajo;
    private double BonoFijo;

    public double calcularSueldo(){
        return this.getañosdeTrabajo()*600 + this.getBonoFijo();
    }
    public int getAñosdeTrabajo() {
        return añosdeTrabajo;
    }

    public void setAñosdeTrabajo(int añosdeTrabajo) {
        this.añosdeTrabajo = añosdeTrabajo;
    }

    public double getBonoFijo() {
        return BonoFijo;
    }

    public void setBonoFijo(double BonoFijo) {
        this.BonoFijo = BonoFijo;
    }
```

Code Smells: Comments

Código antes de aplicar la técnica de Refactorización:

```
//Imprime el listado de estudiantes registrados
public void mostrarListado(){
    //No es necesario implementar
}
```

Consecuencias:

- El simple hecho de mantener un comentario en un método provoca desconfianza en el usuario ya que puede que este método sea difícil de entender o que su nombre no sea apropiado; debe tomarse en cuenta que en algún punto el usuario revisará el código y si encuentra una llamada a dicho método con nombre inapropiado este se ve obligado a revisar la parte donde se encuentra escrito el método solo para saber que función cumple.

Solución usando la técnica de Refactorización: Rename Method

```
public void ListadoEstudiantesRegistrados(){
    //No es necesario implementar
}
```

Code Smells: Speculative Generality

Código antes de aplicar la técnica de Refactorización:

```
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;
```

Consecuencias:

- Analizado el código se aprecia que el atributo facultad no es usado en ninguna parte de este, tener dicho atributo provoca que para un usuario externo sea difícil de entender dicho código por el hecho de que puede pensar que es un atributo que se piensa usar en un futuro.

Solución usando la técnica de Refactorización: Delete Field

```
public class Estudiante{
    //Informacion del estudiante
    public String matricula;
    public String nombre;
    public String apellido;
    public int edad;
    public String direccion;
    public String telefono;
    public ArrayList<Paralelo> paralelos;
```

Code Smells: Duplicate code

Código antes de aplicar la técnica de Refactorización:

```java
//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;
}
```

Consecuencias:

- Ambos métodos realizan la misma función no varía en nada su comportamiento ni los parámetros que recibe.
- Se puede convertir en métodos innecesarios, ya que se puede utilizar cualquiera de los dos y no cambiaría nada.
- El código podría reutilizarse con esto se evita escribir métodos innecesarios.

Solución usando la técnica de Refactorización: Substitute Algorithm y Rename Method

Aplicamos “Substitute Algorithm”, ya que se desea reemplazar un algoritmo existente por uno nuevo que cumpla con ambas funciones y se procederá a eliminar los métodos innecesarios, adicionalmente se debe utilizar “Rename Method”, esto debido a que el método no explicaba lo que hacia por eso hemos decidido colocarle simplemente el nombre de “calcularNota” esto debido a que en un principio había una variable creada dentro del método que decía notaInicial y notaFinal pero no se diferenciaban en nada podemos poner simplemente el nombre de nota a dicha variable.

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

Code Smell: Long Parameter List

Código antes de aplicar las técnicas de Refactorización:

```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;
}
```

Consecuencias:

- Si se mantiene el código tal y como está se incrementa la complejidad del método y es posible que a futuro será difícil de entenderlo, por otro lado, si a futuro se incluyen más parámetros para estos métodos es posible que se vuelva difícil de usar y mantener en cuanto a escalabilidad.

Técnicas usadas: Se implementaron 3 técnicas para resolver este Code Smell, que son: **Remove Parameter**, **Replace Magic Number with Symbolic Constant** y **Move Method**.

Justificación:

Se procedió a sacar los métodos CalcularNotaFinal y CalcularNotaInicial de la clase Estudiante y se los agrego a una nueva clase llamada Notas.java, en esta clase creada por el code smell **Long Parameter List,** contiene los parámetros que entran en los métodos mencionados antes, otro compañero del grupo elimino el code smell **Duplicated Code**, por lo que quedó un solo método el cual calculaba el promedio general de las notas, incluyendo el componente teórico y práctico, luego en la nueva clase Notas.java se procede a crear una referencia al estudiante para saber de quién son las notas, dentro del método calcularPromedio() existían unos valores que representaban los porcentajes de valoración en el componente teórico y práctico, se los removió y se crearon 2 variables con esos valores, todo esto usando el **Replace Magic Number with Symbolic Constant**, sin embargo como aún quedaban los parámetros dentro de los métodos y ya no se usan, se aplica el **Remove Parameter** y con esto queda solucionado el Code Smell.

Código luego de aplicar el refactoring:

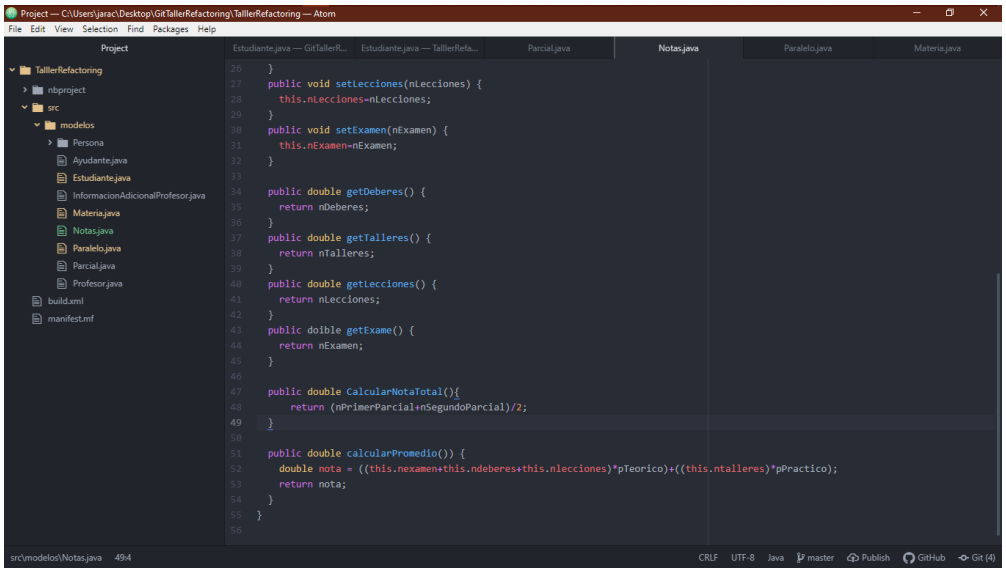
Clase Notas.java

```java

public class Notas {
  private double nDeberes;
  private double nTalleres;
  private double nLecciones;
  private double nExamen;
  private static pTeorico=0.80;
  private static pPractico=0.20;
  public double nPrimerParcial;
  public double nSegundoParcial;
  public double notaTotal;
  private Estudiante est;

  public Notas() {
    nDeberes=0;
    nTalleres=0;
    nLecciones=0;
    nExamen=0;
  }

  public void setDeberes(nDeberes) {
    this.nDeberes=nDeberes;
  }
  public void setTalleres(nTalleres) {
    this.nTalleres=nTalleres;
  }
  public void setLecciones(nLecciones) {
    this.nLecciones=nLecciones;
  }
  public void setExamen(nExamen) {
    this.nExamen=nExamen;
  }
```



Clase Materia.java

```java
package modelos;

public class Materia {
    public String codigo;
    public String nombre;
    public String facultad;

}

```

Clase Estudiante.java

```java

    //Getter y setter de la direccion
    public String getDireccion() {
        return direccion;
    }

    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }

    //Getter y setter del telefono

    public String getTelefono() {
        return telefono;
    }

    public void setTelefono(String telefono) {
        this.telefono = telefono;
    }

}
```