



**Facultad de ingeniería en electricidad y computación**

**Diseño de software**

**Taller 4 segundo parcial**

**Tema: Refactoring**

**Integrantes:**

- ✓ Yepez Vera Tatiana Lisbeth
- ✓ Villagómez Borja Tommy Joel
- ✓ Riera Massache Julián Federico
- ✓ Pita Estrella Bryan Jorge

**Paralelo: 2**

**Profesor: Ingeniero David Jurado**

07/01/2021

PAO 2020 2T

## Tabla de contenido

Refactoring .....	4
Duplicate Code (Code Smell).....	4
a)    Consecuencias: .....	4
b)    Técnica de refactorización: .....	4
c)    Captura código inicial.....	4
d)    Captura código final: .....	5
Long Parameter List (Code Smell) .....	5
a)    Consecuencias: .....	5
b)    Técnica de refactorización: .....	5
c)    Captura código inicial:.....	6
d)    Captura código final: .....	6
Large Class (Code Smell).....	7
a)    Consecuencias: .....	7
b)    Técnica de refactorización: .....	7
c)    Captura código inicial:.....	8
d)    Captura código final: .....	9
Lazy Class (Code Smell) .....	9
a)    Consecuencias: .....	9
b)    Técnica de refactorización: .....	9

c)	Captura código inicial:.....	10
d)	Captura código final: .....	10
	Temporary Field (Code Smell).....	11
a)	Consecuencias: .....	11
b)	Técnica de refactorización: .....	11
c)	Captura código inicial:.....	11
d)	Captura código final: .....	11
	Feature Envy (Code Smell) .....	11
a)	Consecuencias: .....	11
b)	Técnica de refactorización: .....	11
c)	Captura código inicial:.....	12
d)	Captura código final: .....	13
	13	
	Link del repositorio .....	13

## Refactoring

### Duplicate Code (Code Smell)

a) Consecuencias:

Esto puede causar que el código necesite más memoria, de tal manera que lo vuelve mas complejo y a la vez más difícil de entender.

b) Técnica de refactorización:

Para poder solucionar esto podemos usar una técnica de refactorización Extract Method, esta nos va a ayudar a obtener el código duplicado en varios métodos o clases y almacenarlo en un solo método y podemos llamarlo cuando sea necesario, sin tener que volver a escribir demasiado código.

c) Captura código 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

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;
}
```

d) Captura código final:

```
public double CalcularNotaFinal(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres) {
    double notaFinal=0;
    for(Paralelo par:e.getParalelos()) {
        if(p.equals(par)) {
            calcularNota(nexamen, ndeberes, nlecciones, ntalleres);
        }
    }
    return notaFinal;
}

public double calcularNota(double nexamen, double ndeberes, double nlecciones, double ntalleres) {
    double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
    double notaPractico=(ntalleres)*0.20;
    return notaTeorico+notaPractico;
}

//Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. Esta nota es solo el promedio
public double CalcularNotaTotal(Paralelo p) {
    double notaTotal=0;
    for(Paralelo par:e.getParalelos()) {
        if(p.equals(par)) {
            notaTotal=(p.getMateria().notaInicial+p.getMateria().notaFinal)/2;
        }
    }
    return notaTotal;
}
```

## Long Parameter List (Code Smell)

a) Consecuencias:

El usuario puede tener serios problemas al encontrarse con un programa con parámetros muy largos, debido a que tiende a confundir al propio usuario al tener que ingresar mucha información en ese parámetro en particular. Además, puede haber código muerto dentro de dicho parámetro que se hace difícil de identificar ya que su contenido es extenso. También se ve muy mal estéticamente hablando.

b) Técnica de refactorización:

La técnica usada en este caso es la de distribución de responsabilidades. Esto se realiza haciendo un objeto que contenga los parámetros suficientes para hacer la operación dada por un método. De esta manera solo se tendrá que ingresar un parámetro de este tipo de objeto.

c) Captura código inicial:

```
12 public class ObtenerNotas {
13     Estudiante e;
14     //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula
15     public double CalcularNotaInicial(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres) {
16         double notaInicial=0;
17         for(Paralelo par:e.getParalelos()) {
18             if(p.equals(par)) {
19                 calcularNota(nexamen, ndeberes, nlecciones, ntalleres);
20             }
21         }
22         return notaInicial;
23     }
24
25     //Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula
26
27     public double CalcularNotaFinal(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres) {
28         double notaFinal=0;
29         for(Paralelo par:e.getParalelos()) {
30             if(p.equals(par)) {
31                 calcularNota(nexamen, ndeberes, nlecciones, ntalleres);
32             }
33         }
34         return notaFinal;
35     }
36
37     public double calcularNota(double nexamen, double ndeberes, double nlecciones, double ntalleres) {
38         double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
39         double notaPractico=(ntalleres)*0.20;
40         return notaTeorico+notaPractico;
41     }
42 }
```

d) Captura código final:

```
8 /**
9  *
10  * @author Julian
11  */
12 public class Notas {
13     double Nexamen;
14     double Nlecc;
15     double Ntalleres;
16     double Ndeberes;
17     Paralelo pa;
18
19     public Notas(double Nexamen, double Nlecc, double Ntalleres, double Ndeberes, Paralelo pa) {
20         this.Nexamen = Nexamen;
21         this.Nlecc = Nlecc;
22         this.Ntalleres = Ntalleres;
23         this.Ndeberes = Ndeberes;
24     }
25
26     public double getNexamen() {
27         return Nexamen;
28     }
29
30     public void setNexamen(double Nexamen) {
31         this.Nexamen = Nexamen;
32     }
33
34     public double getNlecc() {
35         return Nlecc;
36     }
37
38     public void setNlecc(double Nlecc) {
39         this.Nlecc = Nlecc;
40     }
41
42     public double getNtalleres() {
43         return Ntalleres;
44     }
45
46     public void setNtalleres(double Ntalleres) {
47         this.Ntalleres = Ntalleres;
48     }
49 }
```

```

12 public class ObtenerNotas {
13     Estudiante e;
14     public Notas notas;
15     //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
16     public double CalcularNotaInicial(Notas notas){
17         double notaInicial=0;
18         for(Paralelo par:e.getParalelos()){
19             if(notas.pa.equals(par)){
20                 notaInicial=calcularNota(notas.getNexamen(),notas.getNdeberes(),notas.getNlecc(),notas.getNtalleres());
21             }
22         }
23         return notaInicial;
24     }
25
26     //Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
27
28     public double CalcularNotaFinal(Notas notas){
29         double notaFinal=0;
30         for(Paralelo par:e.getParalelos()){
31             if(notas.pa.equals(par)){
32                 notaFinal=calcularNota(notas.getNexamen(),notas.getNdeberes(),notas.getNlecc(),notas.getNtalleres());
33             }
34         }
35         return notaFinal;
36     }
37     public double calcularNota(double nexamen,double ndeberes, double nlecciones,double ntalleres){
38         double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
39         double notaPractico=(ntalleres)*0.20;
40         return notaTeorico+notaPractico;
41     }
42
43     //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. Esta nota es solo el promedio de las dos calificaciones ant
44     public double CalcularNotaTotal(Paralelo p){
45         double notaTotal=0;
46         for(Paralelo par:e.getParalelos()){
47             if(p.equals(par)){
48                 notaTotal=(p.getMateria().notaInicial+p.getMateria().notaFinal)/2;
49             }
50         }
51         return notaTotal;
52     }

```

## Large Class (Code Smell)

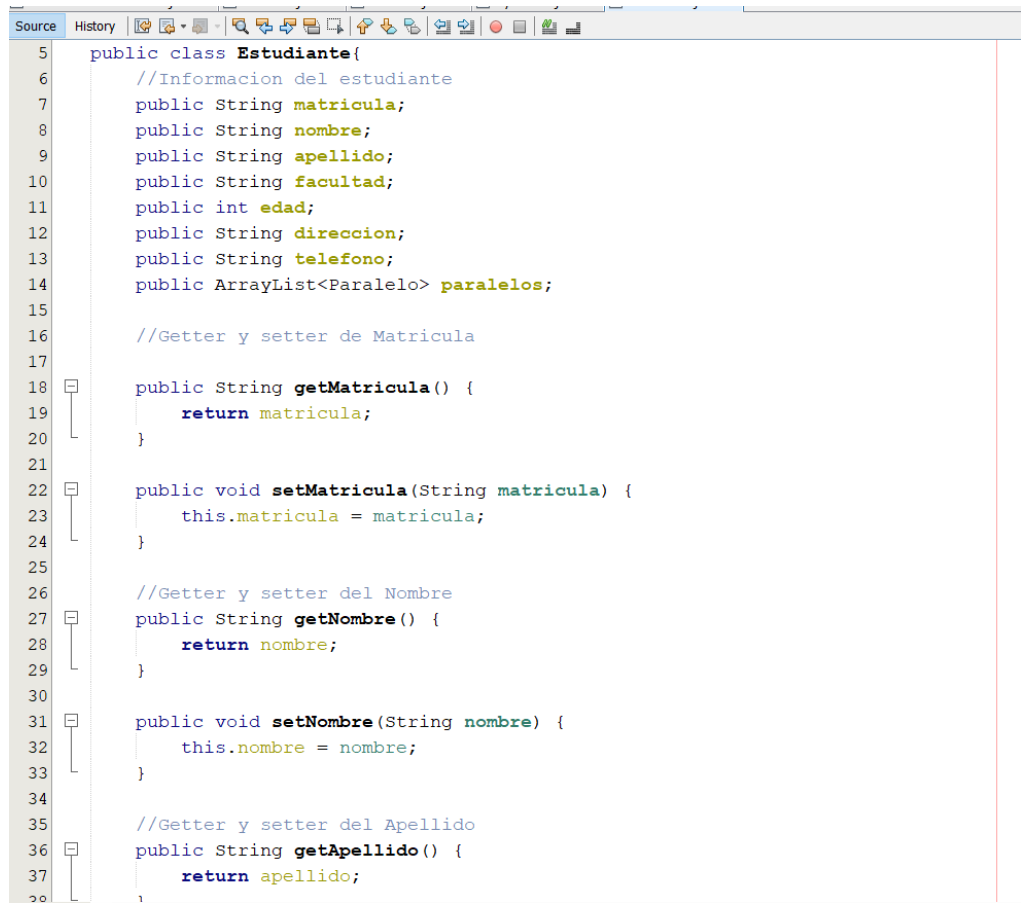
### a) Consecuencias:

La clase es extremadamente grande lo cual hace el código difícil de entender y mantener, además que se viola el principio de Single responsibility al usar métodos para calcular notas en la clase estudiante, lo cual necesariamente debe estar ahí

### b) Técnica de refactorización:

La técnica de factorización que se utilizará para solucionar este code smell es Extract Class, se usó para crear una clase ObtenerNotas la cual contiene los métodos usando para obtener las notas de los estudiante, esto nos ayudara a tener un código más ordenado y fácil de entender.

c) Captura código inicial:



```
5 public class Estudiante{
6     //Informacion del estudiante
7     public String matricula;
8     public String nombre;
9     public String apellido;
10    public String facultad;
11    public int edad;
12    public String direccion;
13    public String telefono;
14    public ArrayList<Paralelo> paralelos;
15
16    //Getter y setter de Matricula
17
18    public String getMatricula() {
19        return matricula;
20    }
21
22    public void setMatricula(String matricula) {
23        this.matricula = matricula;
24    }
25
26    //Getter y setter del Nombre
27    public String getNombre() {
28        return nombre;
29    }
30
31    public void setNombre(String nombre) {
32        this.nombre = nombre;
33    }
34
35    //Getter y setter del Apellido
36    public String getApellido() {
37        return apellido;
38    }
39 }
```



#### d) Captura código final:

```
Source History
12 public class ObtenerNotas {
13     Estudiante e;
14     //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parci
15 public double CalcularNotaInicial(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres){
16     double notaInicial=0;
17     for(Paralelo par:e.getParalelos()){
18         if(p.equals(par)){
19             calcularNota(nexamen, ndeberes, nlecciones, ntalleres);
20         }
21     }
22     return notaInicial;
23 }
24
25 //Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parci
26
27 public double CalcularNotaFinal(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres){
28     double notaFinal=0;
29     for(Paralelo par:e.getParalelos()){
30         if(p.equals(par)){
31             calcularNota(nexamen, ndeberes, nlecciones, ntalleres);
32         }
33     }
34     return notaFinal;
35 }
36
37 public double calcularNota(double nexamen, double ndeberes, double nlecciones, double ntalleres){
38     double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
39     double notaPractico=(ntalleres)*0.20;
40     return notaTeorico+notaPractico;
41 }
42
43 //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. Esta nota es solo el promedio de las dos cal
44 public double CalcularNotaTotal(Paralelo p){
45     double notaTotal=0;
```

### Lazy Class (Code Smell)

#### a) Consecuencias:

Si se mantiene esta clase el proyecto aumenta de tamaño, además de requerir instancias de objetos adicionales para solo tres atributos de tipo primitivos (que están dentro de la clase InformacionAdicionalProfesor()), esto podría generar confusiones al modificar el proyecto en el futuro.

#### b) Técnica de refactorización:

Utilizada Inline class, es decir se eliminan estos atributos de la clase InformacionAdicionalProfesor() añadiéndolos a la clase profesor, lo que permite a su vez eliminar esta primera clase.

c) Captura código inicial:

```
calculaSueloProfesor.java x Profesor.java x Paralelo.java x Ayudante.java x Estudiante.java x InformacionAdicionalProfesor.java x
Source History
1 package modelos;
2
3 public class InformacionAdicionalProfesor {
4     public int añosdeTrabajo;
5     public String facultad;
6     public double BonoFijo;
7
8 }
9
```

d) Captura código final:

```
2
3 import java.util.ArrayList;
4
5 public class Profesor {
6
7     private String codigo;
8     private String nombre;
9     private String apellido;
10    private int edad;
11    private String direccion;
12    private String telefono;
13    private int añosdeTrabajo;
14    private String facultad;
15    private double BonoFijo;
16    private ArrayList<Paralelo> paralelos;
17
18    public Profesor(String codigo, String nombre, String apellido, int edad, String direccion, String telefono, int
19        this.codigo = codigo;
20        this.nombre = nombre;
21        this.apellido = apellido;
22        this.edad = edad;
23        this.direccion = direccion;
24        this.telefono = telefono;
25        this.añosdeTrabajo = añosdeTrabajo;
26        this.facultad = facultad;
27        this.BonoFijo = BonoFijo;
28        this.paralelos = paralelos;
29    }
30
31
32    public Profesor(String codigo, String nombre, String apellido, String facultad, int edad, String direccion, Strin
33        this.codigo = codigo;
```

## Temporary Field (Code Smell)

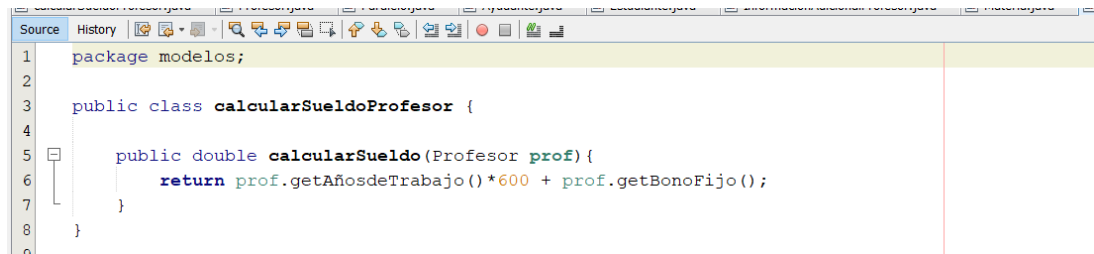
### a) Consecuencias:

Crear una variable compleja para solo un return ocupa espacio innecesario de memoria además de hacer crecer innecesariamente el método en líneas de código

### b) Técnica de refactorización:

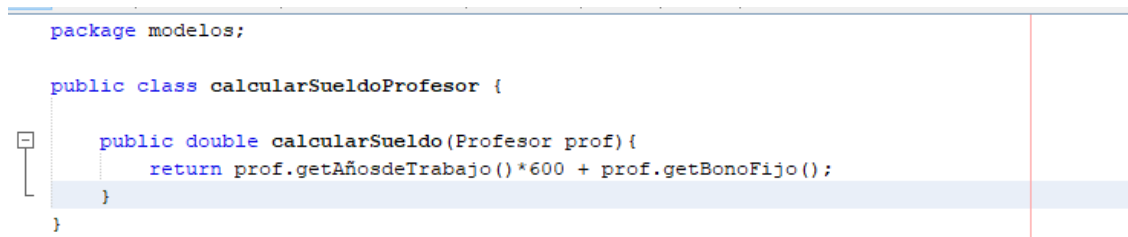
Se usa inline temp, de esta forma se elimina la declaración y la inicialización de la variable sueldo, y directamente retornara la expresión (tener en cuenta que se cambia la expresión ya que se eliminó el objeto tipo info de la clase InformacionAdicionalProfesor() y se utilizó directamente los getter de los atribuidos de esta clase, debido al método de refactorización aplicado anteriormente(inline class)

### c) Captura código inicial:



```
1 package modelos;
2
3 public class calcularSueltoProfesor {
4
5     public double calcularSuelto(Profesor prof){
6         return prof.getAñosdeTrabajo()*600 + prof.getBonoFijo();
7     }
8 }
9
```

### d) Captura código final:



```
package modelos;

public class calcularSueltoProfesor {

    public double calcularSuelto(Profesor prof){
        return prof.getAñosdeTrabajo()*600 + prof.getBonoFijo();
    }
}
```

## Feature Envy (Code Smell)

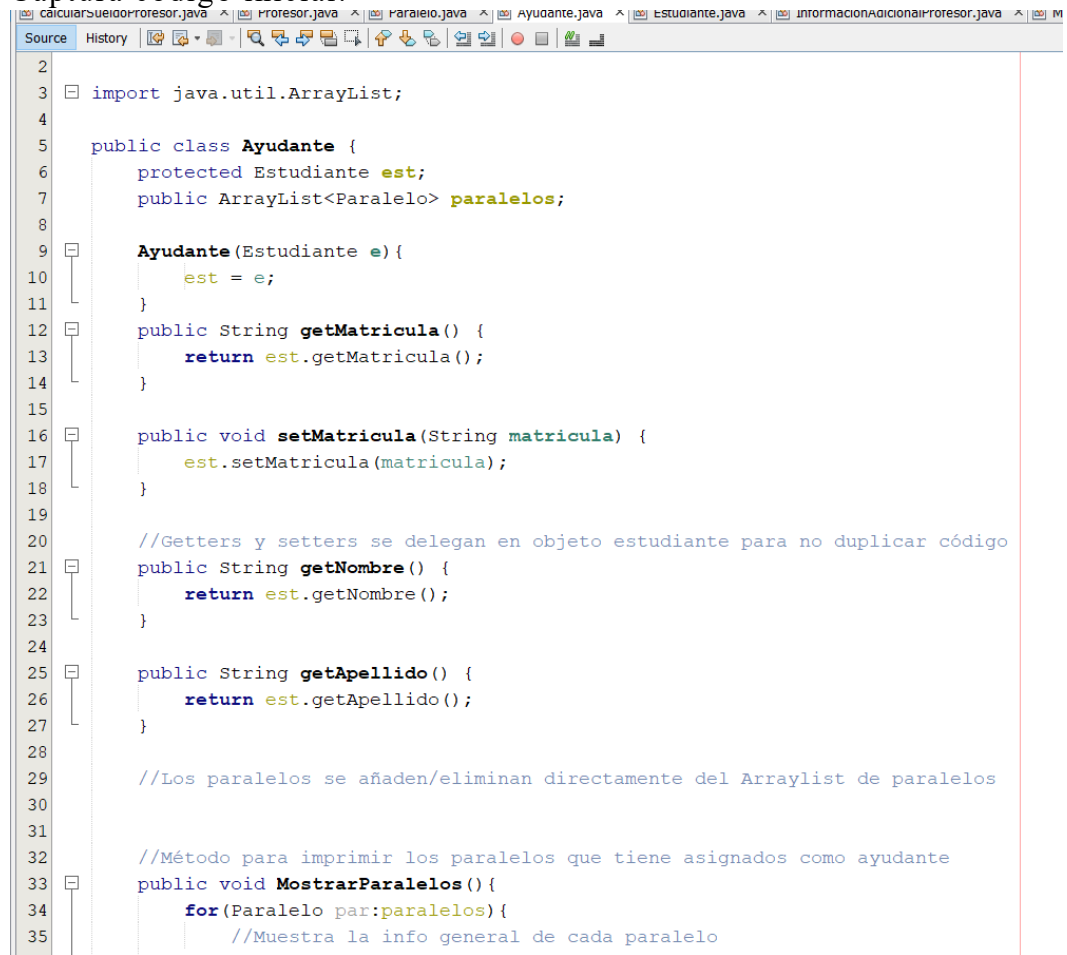
### a) Consecuencias:

En el código inicial se tiene 2 clases diferentes que tienen código parecido, se puede notar que entre estas clases existe un alto acoplamiento, si se elimina o cambia la clase estudiante esto afectará a la clase ayudante.

### b) Técnica de refactorización:

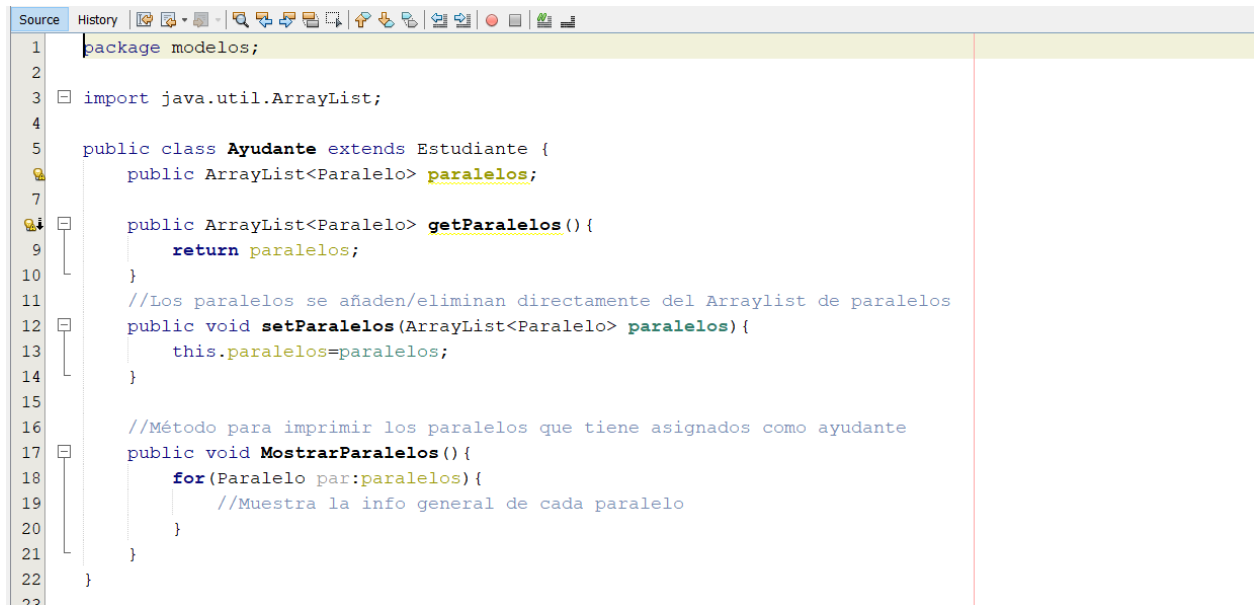
La técnica aplicada para resolver este code smell es Reemplazar delegación con herencia, ya que a través de herencia se puede hacer que el código en la clase estudiante sea reutilizado por la clase ayudante.

c) Captura código inicial:



```
2
3 import java.util.ArrayList;
4
5 public class Ayudante {
6     protected Estudiante est;
7     public ArrayList<Paralelo> paralelos;
8
9     Ayudante(Estudiante e) {
10         est = e;
11     }
12     public String getMatricula() {
13         return est.getMatricula();
14     }
15
16     public void setMatricula(String matricula) {
17         est.setMatricula(matricula);
18     }
19
20     //Getters y setters se delegan en objeto estudiante para no duplicar código
21     public String getNombre() {
22         return est.getNombre();
23     }
24
25     public String getApellido() {
26         return est.getApellido();
27     }
28
29     //Los paralelos se añaden/eliminan directamente del ArrayList de paralelos
30
31
32     //Método para imprimir los paralelos que tiene asignados como ayudante
33     public void MostrarParalelos() {
34         for (Paralelo par:paralelos){
35             //Muestra la info general de cada paralelo
```

d) Captura código final:



```
1 package modelos;
2
3 import java.util.ArrayList;
4
5 public class Ayudante extends Estudiante {
6     public ArrayList<Paralelo> paralelos;
7
8     public ArrayList<Paralelo> getParalelos() {
9         return paralelos;
10    }
11    //Los paralelos se añaden/eliminan directamente del ArrayList de paralelos
12    public void setParalelos(ArrayList<Paralelo> paralelos) {
13        this.paralelos=paralelos;
14    }
15
16    //Método para imprimir los paralelos que tiene asignados como ayudante
17    public void MostrarParalelos() {
18        for(Paralelo par:paralelos) {
19            //Muestra la info general de cada paralelo
20        }
21    }
22 }
```

## Link del repositorio

<https://github.com/Jfed98/TallerRefactoring>