



TALLER DE REFACTORING

Encontrar tipos de Code Smells y solucionarlos

Descripción breve:

En equipos de trabajo, conformados por hasta tres estudiantes, (i) identifique los malos olores de programación encontrados en el código fuente adjunto, (ii) Identifique las técnicas de refactorización adecuadas para eliminar los malos olores encontrados y (iii) Refactorice el código fuente para obtener un código más limpio y fácil de leer. Justifique su respuesta.

INTEGRANTES:

- Benalcázar García Sebastián Elías
- Poveda Páez Alexis Steven
- Townsend Hinostroza Darinka Milena

CONTENIDO

OBJETIVOS ESPECÍFICOS	3
DESCRIPCIÓN	3
ESPECIFICACIONES	3
Sección A	3
Sección B	3
SECCION A - CODE SMELLS IDENTIFICADOS	4
1. Code Smells #1: Duplicate Code	4
1. Consecuencia:	4
2. Solución:	4
2. Code Smells #2: Clase Perezosa	5
1. Consecuencia:	5
2. Solución:	5
3. Code Smells #3: Dead Code	6
1. Consecuencia:	6
2. Solución:	6
4. Code Smells #3: Data Class	7
1. Consecuencia:	7
2. Solución:	7
5. Code Smells #5: Speculate Generality	8
1. Consecuencia:	8
2. Solución:	8
6. Code Smells #6: Feature Envy	9
1. Consecuencia:	9
2. Solución:	9
7. Code Smells #7: Comments	10
1. Consecuencia:	10
2. Solución:	10
8. Code Smells #8: Long Class	11
1. Consecuencia:	11
2. Solución:	11

OBJETIVOS ESPECÍFICOS

- Identificar malos olores de programación en el código fuente adjunto y las técnicas de refactorización correspondientes.
- Aplicar técnicas de refactorización que pueden aplicarse para eliminar los malos olores previamente identificados.

DESCRIPCIÓN

En equipos de trabajo, conformados por hasta tres estudiantes, (i) identifique los malos olores de programación encontrados en el código fuente adjunto, (ii) Identifique las técnicas de refactorización adecuadas para eliminar los malos olores encontrados y (iii) Refactorice el código fuente para obtener un código más limpio y fácil de leer. Justifique su respuesta.

ESPECIFICACIONES

Considere un **sistema académico** que permite manejar los estudiantes registrados en ciertos paralelos de distintas materias. Además, cada materia tiene un profesor y podría tener asignado un ayudante.

Sección A

Elabore un **reporte** en el que **identifique los code smells** encontrados en el código adjunto. Para cada code smell debe **indicar el nombre**, las **consecuencias** de mantener el mismo en dicho código y la(s) técnicas de refactorización utilizadas para eliminarlo. Para cada mal olor coloque una captura inicial y una del código refactorizado. Indique cualquier asunción que realice. [60%]

Sección B

Cree un repositorio en GitHub para subir el reporte y realizar las mejoras sobre el código inicial y aplique las técnicas indicadas en el reporte. [40%]

SECCION A - CODE SMELLS IDENTIFICADOS

1. Code Smells #1: Duplicate Code

1. Consecuencia:

Se puede identificar Duplicate Code en las clases Profesor y Estudiante, puesto que varios de sus atributos se repiten, como los atributos nombre, apellido, edad y dirección.

```
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,
        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);
    }
}

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;
    }
}
```

2. Solución:

Para solucionar este code smell se utilizó extract method, creamos una nueva clase persona donde almacenará los atributos y métodos que se repetían en las clases profesor, estudiante y ayudante.

```
public class Persona {
    public String nombre;
    public String apellido;
    public int edad;

    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;
    }

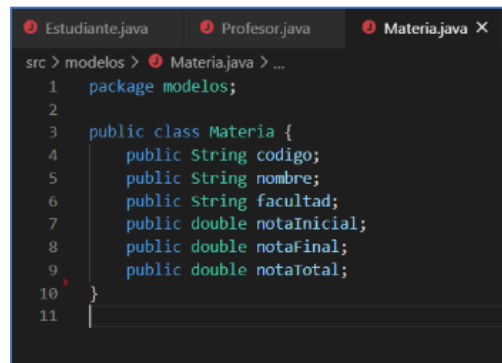
    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```

2. Code Smells #2: Clase Perezosa

1. Consecuencia:

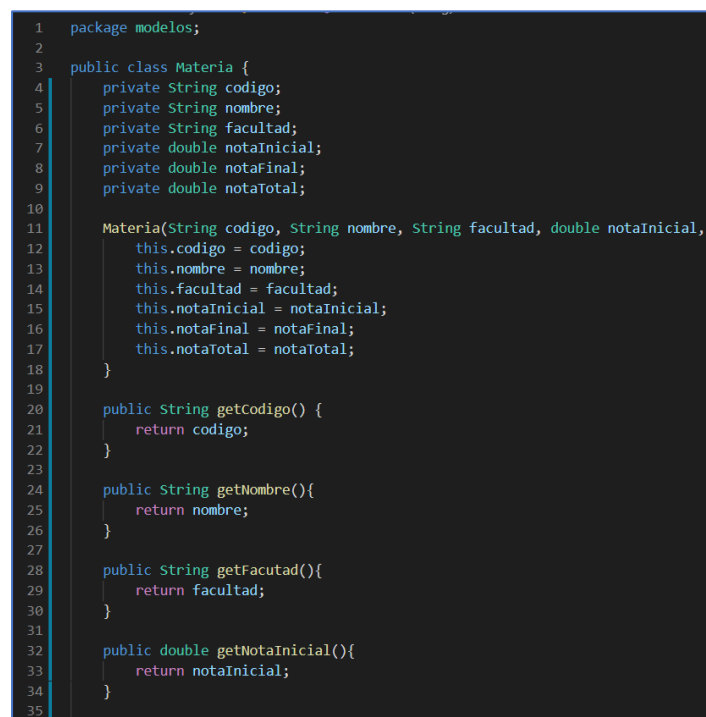
Toda clase tiene un costo por ser mantenida y leída así sea mínima su construcción. La clase materia en este caso no cuenta con un constructor o métodos para poder instanciar sus atributos y obtener funcionamiento alguno dentro del sistema.



```
src > modelos > Materia.java > ...
1  package modelos;
2
3  public class Materia {
4      public String codigo;
5      public String nombre;
6      public String facultad;
7      public double notaInicial;
8      public double notaFinal;
9      public double notaTotal;
10 }
11
```

2. Solución:

Para refactorizar el código de la clase perezosa, es necesario destruir herencias en el caso de que existan o si no existe una relación en línea, eliminarla. En esta ocasión al denotar la importancia de la clase Materia, se le dio más peso a la clase.



```
1  package modelos;
2
3  public class Materia {
4      private String codigo;
5      private String nombre;
6      private String facultad;
7      private double notaInicial;
8      private double notaFinal;
9      private double notaTotal;
10
11      Materia(String codigo, String nombre, String facultad, double notaInicial,
12              double notaFinal, double notaTotal) {
13          this.codigo = codigo;
14          this.nombre = nombre;
15          this.facultad = facultad;
16          this.notaInicial = notaInicial;
17          this.notaFinal = notaFinal;
18          this.notaTotal = notaTotal;
19      }
20
21      public String getCodigo() {
22          return codigo;
23      }
24
25      public String getNombre(){
26          return nombre;
27      }
28
29      public String getFacultad(){
30          return facultad;
31      }
32
33      public double getNotaInicial(){
34          return notaInicial;
35      }
36
37      public double getNotaFinal(){
38          return notaFinal;
39      }
40
41      public double getNotaTotal(){
42          return notaTotal;
43      }
44
45 }
```

3. Code Smells #3: Dead Code

1. Consecuencia:

Existe código sin utilizar dentro de las clases. En la clase `Materia` existen varios atributos no instanciados, al igual que en la clase `Paralelo` existe un método llamado `mostrarListado()` el cual no cuenta con implementación

<pre>public class Materia { public String codigo; public String nombre; public String facultad; public double notaInicial; public double notaFinal; public double notaTotal; }</pre>	<pre>//Imprime el listado de estudiantes registrados public void mostrarListado(){ //No es necesario implementar }</pre>
--	--

2. Solución:

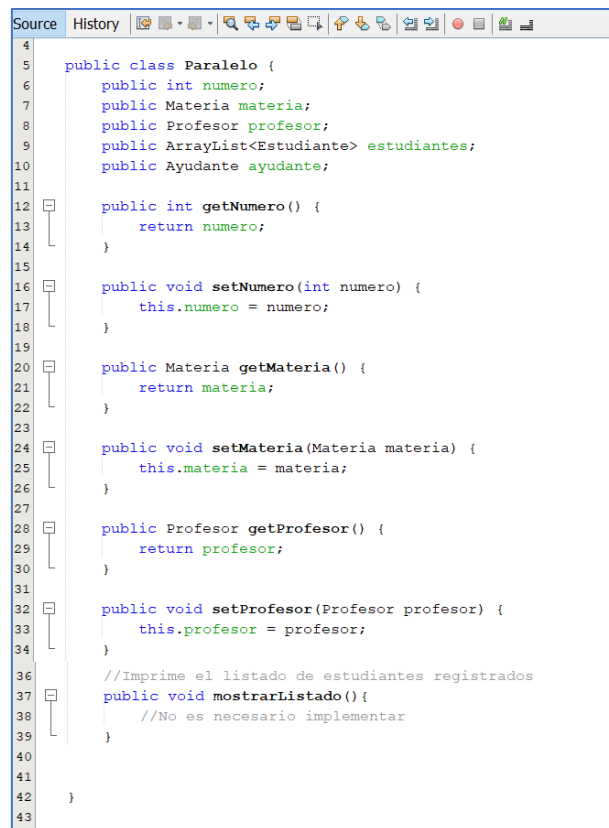
Se añadió el código necesario para que `Materia` tenga más responsabilidad y se eliminó el método `mostrarListado()` que no tenía implementación.

<pre>4 5 public class Paralelo { 6 public int numero; 7 public Materia materia; 8 public Profesor profesor; 9 public ArrayList<Estudiante> estudiantes; 10 public Ayudante ayudante; 11 12 public int getNumero() { 13 return numero; 14 } 15 16 public void setNumero(int numero) { 17 this.numero = numero; 18 } 19 20 public Materia getMateria() { 21 return materia; 22 } 23 24 public void setMateria(Materia materia) { 25 this.materia = materia; 26 } 27 28 public Profesor getProfesor() { 29 return profesor; 30 } 31 32 public void setProfesor(Profesor profesor) { 33 this.profesor = profesor; 34 } 35 36 } 37</pre>	<pre>1 package modelos; 2 3 public class Materia { 4 private String codigo; 5 private String nombre; 6 private String facultad; 7 private double notaInicial; 8 private double notaFinal; 9 private double notaTotal; 10 11 Materia(String codigo, String nombre, String facultad, double notaInicial, 12 double notaFinal, double notaTotal) { 13 this.codigo = codigo; 14 this.nombre = nombre; 15 this.facultad = facultad; 16 this.notaInicial = notaInicial; 17 this.notaFinal = notaFinal; 18 this.notaTotal = notaTotal; 19 } 20 21 public String getcodigo() { 22 return codigo; 23 } 24 25 public String getNombre(){ 26 return nombre; 27 } 28 29 public String getFacutad(){ 30 return facultad; 31 } 32 33 public double getNotaInicial(){ 34 return notaInicial; 35 } 36 37 }</pre>
--	---

4. Code Smells #3: Data Class

1. Consecuencia:

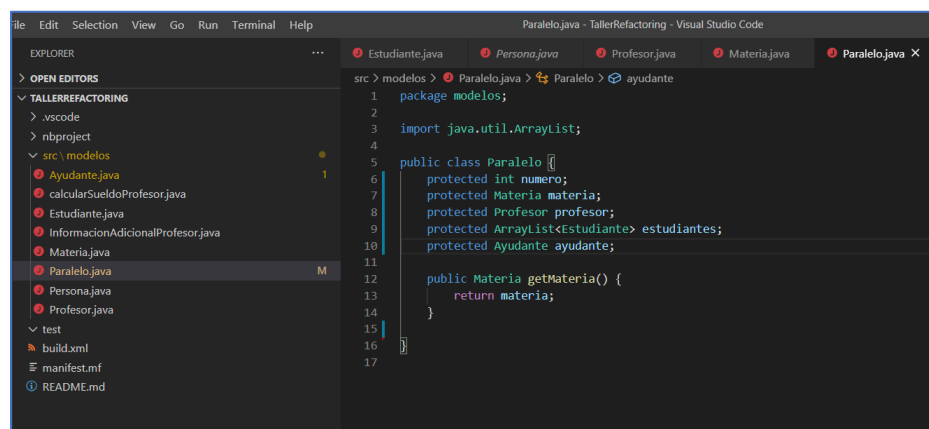
El Code Smell Data class se puede identificar dentro de la clase Paralelo, puesto que esta clase solo tiene getters y setters, además de un método llamado mostrarParelo() que no es utilizado.



```
4
5 public class Paralelo {
6     public int numero;
7     public Materia materia;
8     public Profesor profesor;
9     public ArrayList<Estudiante> estudiantes;
10    public Ayudante ayudante;
11
12    public int getNumero() {
13        return numero;
14    }
15
16    public void setNumero(int numero) {
17        this.numero = numero;
18    }
19
20    public Materia getMateria() {
21        return materia;
22    }
23
24    public void setMateria(Materia materia) {
25        this.materia = materia;
26    }
27
28    public Profesor getProfesor() {
29        return profesor;
30    }
31
32    public void setProfesor(Profesor profesor) {
33        this.profesor = profesor;
34    }
35
36    //Imprime el listado de estudiantes registrados
37    public void mostrarListado() {
38        //No es necesario implementar
39    }
40
41 }
42
43
```

2. Solución:

Se utilizó el método de refactorización Encapsulate Field. Para solucionar este problema se encapsularon los atributos cambiando su estado a privados.



```
Paralelo.java - TallerRefactoring - Visual Studio Code
src > modelos > Paralelo.java > Paralelo > ayudante
1 package modelos;
2
3 import java.util.ArrayList;
4
5 public class Paralelo {
6     protected int numero;
7     protected Materia materia;
8     protected Profesor profesor;
9     protected ArrayList<Estudiante> estudiantes;
10    protected Ayudante ayudante;
11
12    public Materia getMateria() {
13        return materia;
14    }
15
16 }
17
```

5. Code Smells #5: Speculate Generality

1. Consecuencia:

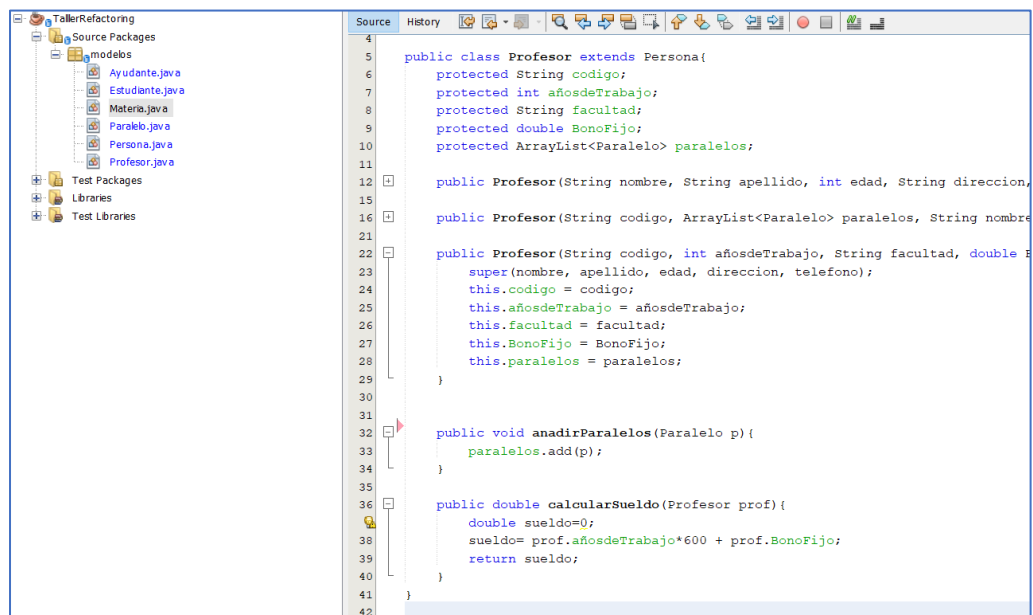
Existe código que busca cubrir posibles casos futuros que no necesariamente puedan llegar a ocurrir. En la clase InformaciónAdicionalProfesor, existe un caso parecido, el nombre de la clase hacer referencia al posible caso de que se necesite agregar información adicional no requerida para los objetos de tipo Profesor.

```
package modelos;

public class InformaciónAdicionalProfesor {
    public int añosdeTrabajo;
    public String facultad;
    public double BonoFijo;
}
```

2. Solución:

Para este caso en específico en donde la clase informaciónAdicional, posee dead code, es considerado una clase vaga y a su vez forma parte de una Speculate Generality la mejor forma de refactorizar fue Safely delete, tanto de la clase informaciónAdicional como la clase calcularSueldoProfesor moviendo sus atributos y método (respectivamente) a la clase profesor y generando un constructor para el caso de recibir la información adicional utilizada en el método calcular sueldo.



6. Code Smells #6: Feature Envy

1. Consecuencia:

Se logra identificar Feature Envy en la clase Ayudante, puesto esta clase utiliza los métodos y atributos de la clase Estudiante mucho más que sus propios métodos y atributos.

```
1 package modelos;
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     //Método para imprimir los paralelos que tiene asignados como ayudante
```

2. Solución:

Se extendió la clase Ayudante a Estudiante para que use los métodos correspondientes de Estudiante y no envíe sus métodos, ya que anteriormente usaba más los métodos de Estudiante que la misma.

```
Gu Run Terminal Help Ayudante.java - TallerRefactoring - Visual Studio Code
src > modelos > Ayudante.java > ...
1 package modelos;
2
3 import java.util.ArrayList;
4
5 public class Ayudante extends Estudiante{
6     protected ArrayList<Paralelo> paralelosAyudante;
7
8     public Ayudante(String matricula, String facultad, ArrayList<Paralelo> paralelos, String nombre, String apellido, int edad, String direccion, String telefono) {
9         super(matricula, facultad, paralelos, nombre, apellido, edad, direccion, telefono);
10        this.paralelosAyudante = new ArrayList<Paralelo>();
11    }
12
13    public void ImprimirParalelosDelAyudante(){
14        for(Paralelo par: paralelosAyudante){
15            //Muestra la info general de cada paralelo
16        }
17    }
18
19 }
```

7. Code Smells #7: Comments

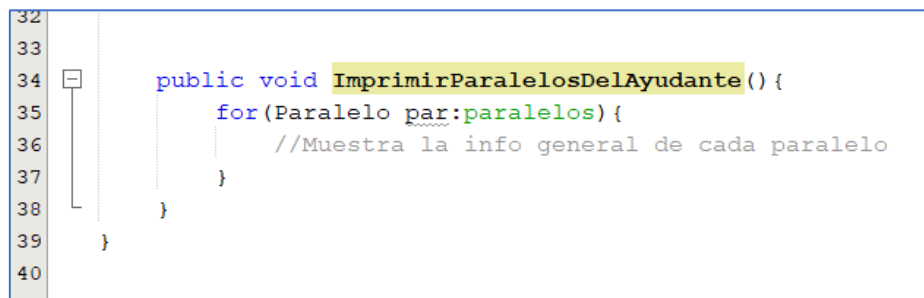
1. Consecuencia:

Se puede identificar el Code Smell comments en la clase Ayudante. Los comentarios realizados ayudan a que el programados pueda entenderse, lo que no debería ser así puesto que el código debe ser intuitivo y no necesitar de códigos.



2. Solución:

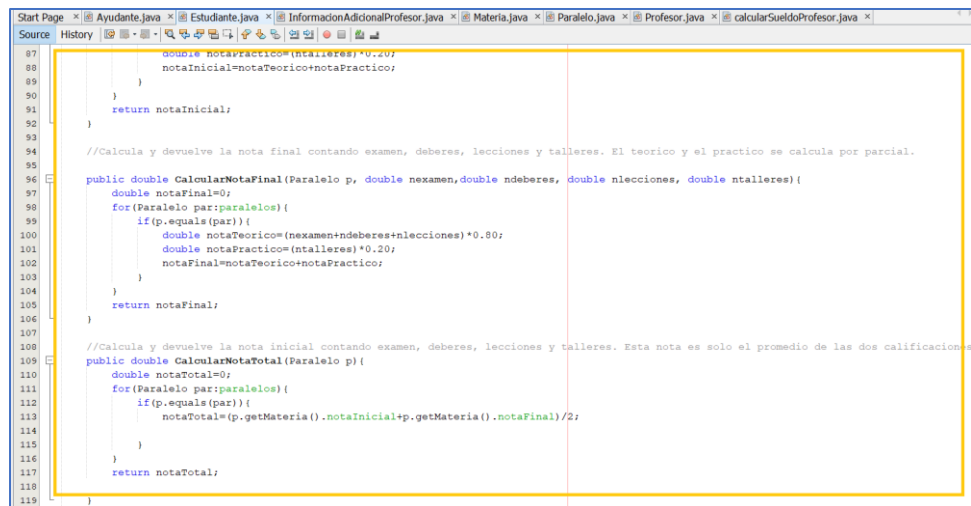
Para solucionar este Code Smell se debe utilizar el tipo refactoring Rename Method, así el nombre del método puede dar a entender al programador para que va a servir.



8. Code Smells #8: Long Class

1. Consecuencia:

Dentro de la clase Estudiante se puede identificar el Code Smell Long class, esta clase es muy larga, tiene muchos métodos y trata de hacer muchas acciones las cuales algunas pueden ser realizadas por otra clase.



```
87     double notaPractico=(ntalleres)*0.20;
88     notaInicial=notaTeorico+notaPractico;
89
90 }
91 return notaInicial;
92 }
93
94 //Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
95
96 public double CalcularNotaFinal(Paralelo p, double nexamen,double ndeberes, double nlecciones, double ntalleres){
97     double notaFinal=0;
98     for(Paralelo par:paralelos){
99         if(p.equals(par)){
100             double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
101             double notaPractico=(ntalleres)*0.20;
102             notaFinal=notaTeorico+notaPractico;
103         }
104     }
105     return notaFinal;
106 }
107
108 //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. Esta nota es solo el promedio de las dos calificaciones
109 public double CalcularNotaTotal(Paralelo p){
110     double notaTotal=0;
111     for(Paralelo par:paralelos){
112         if(p.equals(par)){
113             notaTotal=(p.getMateria().notaInicial+p.getMateria().notaFinal)/2;
114         }
115     }
116     return notaTotal;
117 }
118
119 }
```

2. Solución:

Para solucionar este Code Smell se puede utilizar el refactoring llamado Extract Class, la cual separa la clase Estudiante en dos, en este caso se creó una nueva clase Persona que almacena varios métodos y atributos de estudiantes, haciendo que la clase Estudiante sea más pequeña.

```

Start Page x Ayudante.java x Estudiante.java x Materia.java x Paralelo.java x Profesor.java x Pr
Source History
1 package modelos;
2
3 import java.util.ArrayList;
4
5 public class Estudiante extends Persona{
6     //Informacion del estudiante
7     public String matricula;
8     public String facultad;
9     public ArrayList<Paralelo> paralelos;
10
11     public Estudiante(String matricula, String facultad, ArrayList<Paralelo> pa
12         super(nombre, apellido, edad, direccion, telefono);
13         this.matricula = matricula;
14         this.facultad = facultad;
15         this.paralelos = paralelos;
16     }
17
18     //Getter y setter de Matricula
19
20     public String getMatricula() {
21         return matricula;
22     }
23
24     public void setMatricula(String matricula) {
25         this.matricula = matricula;
26     }
27
28     //Getter y setter de la Facultad
29     public String getFacultad() {
30         return facultad;
31     }
32
33     //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y t
34     public double CalcularNotaInicial(Paralelo p, double nexamen, double ndeberes,
35         double notaInicial=0;
36         for(Paralelo par:paralelos){
37             if(p.equals(par)){
38                 double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
39                 double notaPractico=(ntalleres)*0.20;
40                 notaInicial=notaTeorico+notaPractico;
41             }
42         }
43         return notaInicial;
44     }
45
46     //Calcula y devuelve la nota final contando examen, deberes, lecciones y tal
47     public double CalcularNotaFinal(Paralelo p, double nexamen, double ndeberes,
48         double notaFinal=0;
49         for(Paralelo par:paralelos){
50             if(p.equals(par)){
51                 double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
52                 double notaPractico=(ntalleres)*0.20;
53                 notaFinal=notaTeorico+notaPractico;
54             }
55         }
56         return notaFinal;
57     }
58
59     //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y t
60     public double CalcularNotaTotal(Paralelo p){
61         double notaTotal=0;
62     }
63
64
65
66

```