

# Taller 09

# Code Smells

Estudiantes:

- Karla Durán Oscuez
- Bryan Plaza Anchundia
- Betsy Nazareno Aguiño

Profesor:

Msc. David Jurado Mosquera

Curso:

Diseño de Software

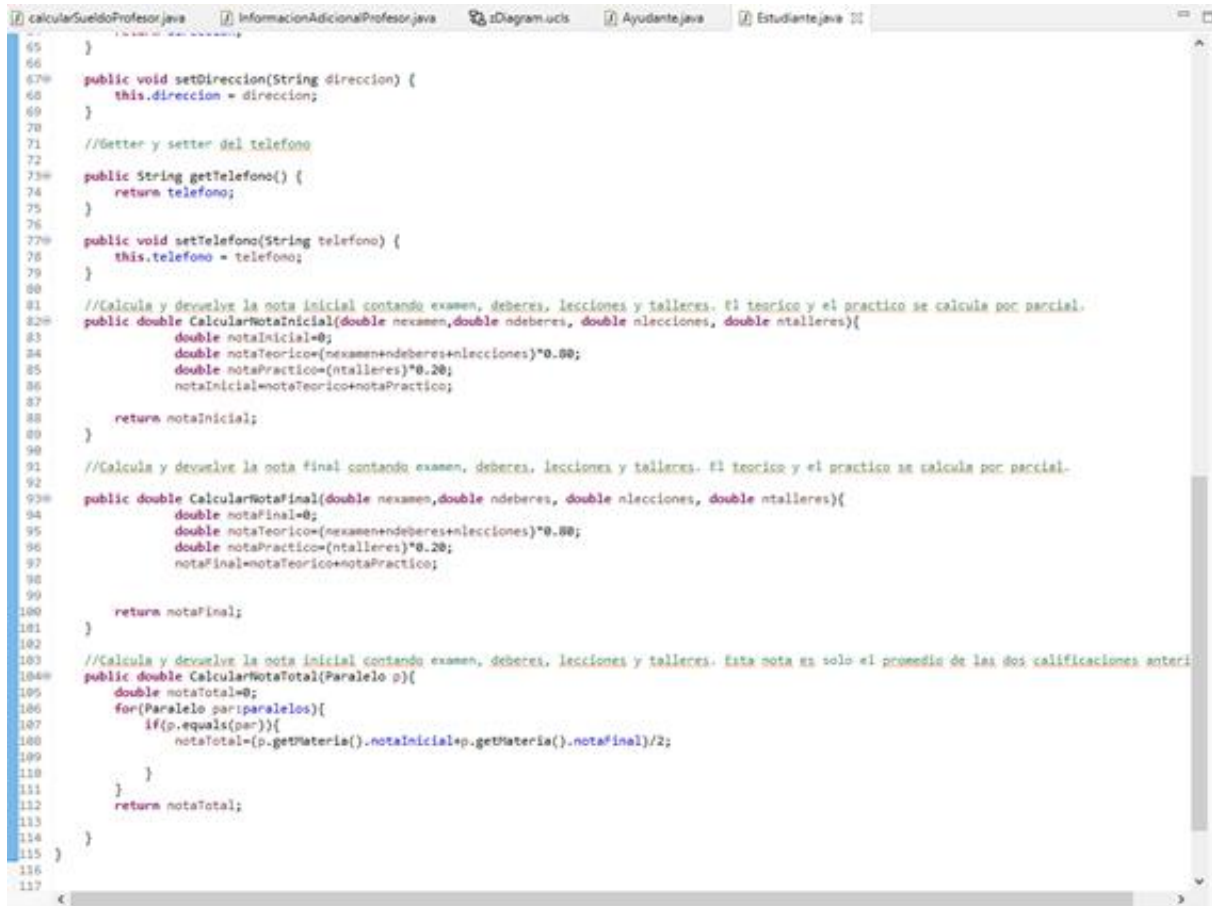


## Contenido

Code Smell: Long Class .....	3
Código inicial.....	3
Consecuencias y técnicas de refactorización .....	3
Código modificado.....	4
Code Smell: Long Parameter List.....	5
Código inicial / code smell .....	5
Consecuencias y técnicas de refactorización .....	5
Código modificado.....	6
Code Smell: Feature Envy .....	7
Código inicial / code smell .....	7
Consecuencias y técnicas de refactorización .....	7
Código modificado.....	7
Code Smell: Data class .....	8
Código inicial / code smell .....	8
Consecuencias y técnicas de refactorización .....	8
Código modificado.....	9
Code Smell: Duplicate code .....	10
Código inicial / code smell .....	10
Consecuencias y técnicas de refactorización .....	10
Código modificado.....	11
Code Smell: Inapropriate intimacy.....	12
Código inicial / code smell .....	12
Consecuencias y técnicas de refactorización .....	12
Código modificado.....	13

# Code Smell: Long Class

## Código inicial



```
65 }
66
67 public void setDireccion(String direccion) {
68     this.direccion = direccion;
69 }
70
71 //Getter y setter del telefono
72
73 public String getTelefono() {
74     return telefono;
75 }
76
77 public void setTelefono(String telefono) {
78     this.telefono = telefono;
79 }
80
81 //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
82 public double CalculaNotaInicial(double nexamen, double ndeberes, double nlecciones, double ntalleres) {
83     double notaInicial = 0;
84     double notaTeorico = (nexamen + ndeberes + nlecciones) * 0.80;
85     double notaPractico = (ntalleres) * 0.20;
86     notaInicial = notaTeorico + notaPractico;
87
88     return notaInicial;
89 }
90
91 //Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
92
93 public double CalculaNotaFinal(double nexamen, double ndeberes, double nlecciones, double ntalleres) {
94     double notaFinal = 0;
95     double notaTeorico = (nexamen + ndeberes + nlecciones) * 0.80;
96     double notaPractico = (ntalleres) * 0.20;
97     notaFinal = notaTeorico + notaPractico;
98
99     return notaFinal;
100 }
101
102 //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. Esta nota es solo el promedio de las dos calificaciones anteriores.
103 public double CalculaNotaTotal(Paralelo p) {
104     double notaTotal = 0;
105     for (Paralelo par : paralelos) {
106         if (p.equals(par)) {
107             notaTotal = (p.getMateria().getMateria().notaInicial + p.getMateria().notaFinal) / 2;
108         }
109     }
110
111     return notaTotal;
112 }
113
114 }
115
116
117
```

## Consecuencias y técnicas de refactorización

La clase Estudiante del proyecto presenta el code smell Long Class, puesto que contiene muchos métodos, concluyendo en muchas líneas de código. Se ha optado por extraer una clase, de nombre CalculodeNotas esto porque los métodos CalculaNotaInicial, CalculaNotaFinal y CalculaNotaTotal no deberían pertenecer a la clase Estudiante, hay una independencia, además de que la clase se libera de responsabilidades que no le corresponden, cumpliendo el principio SOLID de Single Responsibility.

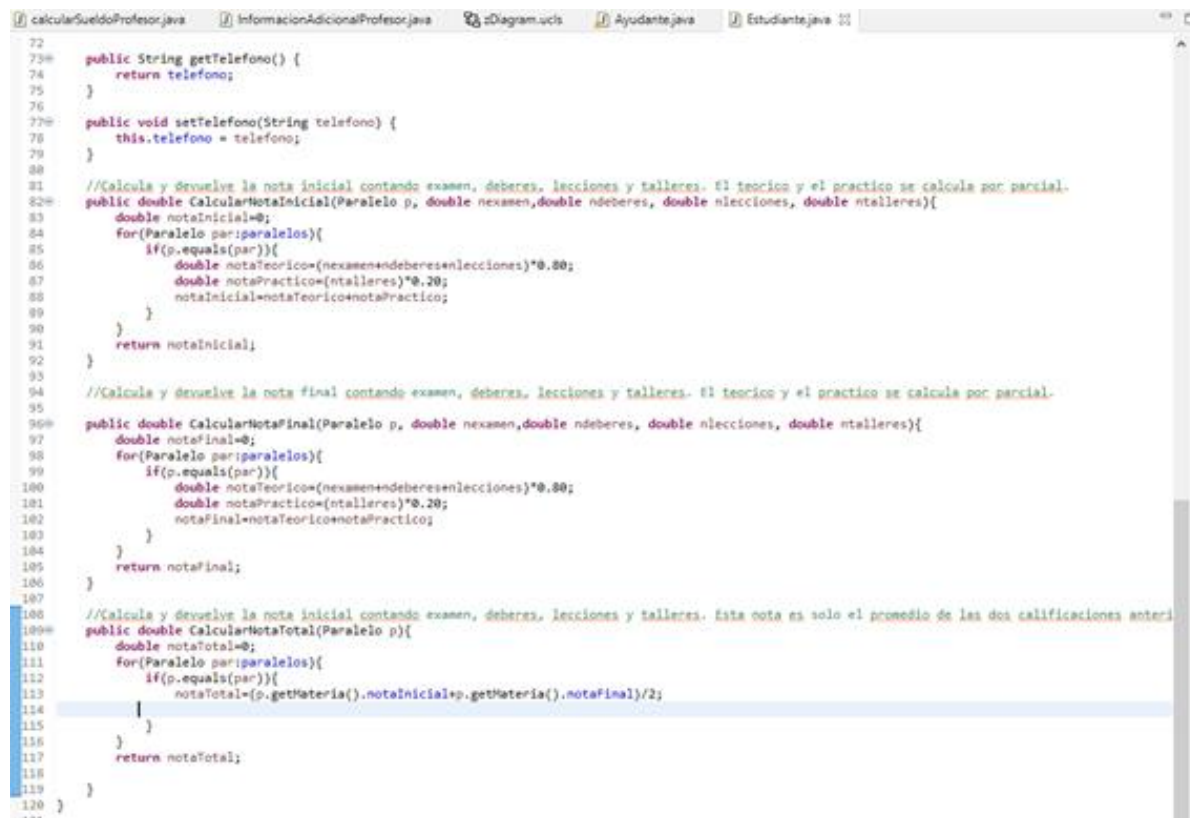
## Código modificado

```
calculaSueloProfesor.java  InformaciónAdicionalProfesor.java  zDiagram.ucs  Ayudante.java  Estudiante.java  CalculoDeNotas.java
1  package modelo;
2
3  public class Estudiante {
4      //Información del estudiante
5      public String matricula;
6      public String nombre;
7      public String apellido;
8      public String facultad;
9      public int edad;
10     public String direccion;
11     public String telefono;
12     public ArrayList<Paralelo> paralelos;
13
14     //Getter y setter de Matricula
15
16     public String getMatricula() {
17         return matricula;
18     }
19
20     public void setMatricula(String matricula) {
21         this.matricula = matricula;
22     }
23
24     //Getter y setter del Nombre
25
26     public String getNombre() {
27         return nombre;
28     }
29
30     public void setNombre(String nombre) {
31         this.nombre = nombre;
32     }
33
34     //Getter y setter del Apellido
35
36     public String getApellido() {
37         return apellido;
38     }
39
40     public void setApellido(String apellido) {
41         this.apellido = apellido;
42     }
43
44     //Getter y setter de la Facultad
45
46     public String getFacultad() {
47         return facultad;
48     }
49
50     public void setFacultad(String facultad) {
51         this.facultad = facultad;
52     }
53
54     //Getter y setter de la edad
55
56     public int getEdad() {
57         return edad;
58     }
59 }
60
```

```
calculaSueloProfesor.java  InformaciónAdicionalProfesor.java  zDiagram.ucs  Ayudante.java  Estudiante.java  CalculoDeNotas.java
1  package modelo;
2
3  public class CalculoDeNotas {
4
5
6      //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
7      public static double CalcularNotaInicial(double nexamen, double ndeberes, double nlecciones, double ntalleres) {
8
9          double notaInicial = 0;
10         double notaTeorico = (nexamen + ndeberes + nlecciones) * 0.80;
11         double notaPractico = (ntalleres) * 0.20;
12         notaInicial = notaTeorico + notaPractico;
13
14         return notaInicial;
15     }
16
17     //Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
18
19     public static double CalcularNotaFinal(double nexamen, double ndeberes, double nlecciones, double ntalleres) {
20
21         double notaFinal = 0;
22         double notaTeorico = (nexamen + ndeberes + nlecciones) * 0.80;
23         double notaPractico = (ntalleres) * 0.20;
24         notaFinal = notaTeorico + notaPractico;
25
26         return notaFinal;
27     }
28
29     //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. Esta nota es solo el promedio de las dos calificaciones anterior
30     public static double CalcularNotaTotal(Paralelo p) {
31         double notaTotal = 0;
32
33         notaTotal = (p.getMateria().notaInicial + p.getMateria().notaFinal) / 2;
34
35         return notaTotal;
36     }
37 }
38
39 }
40
```

# Code Smell: Long Parameter List

## Código inicial / code smell



```
72
73 public String gettelefono() {
74     return telefono;
75 }
76
77 public void settelefono(String telefono) {
78     this.telefono = telefono;
79 }
80
81 //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
82 public double CalcularNotaInicial(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres){
83     double notaInicial=0;
84     for(Paralelo par:paralelos){
85         if(p.equals(par)){
86             double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
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 anteri
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 }
120 }
```

## Consecuencias y técnicas de refactorización

El code smell Long Parameter List en el presente proyecto tiene lugar en la clase `CalculodeNotas` clase creada de extraer métodos de la clase `Estudiante`, en específico en los métodos `CalcularNotaInicial` y `CalcularNotaFinal`. Por definición este code smell implica tener más de tres o cuatro parámetros para un determinado método, en los mencionados anteriormente hay cinco parámetros.

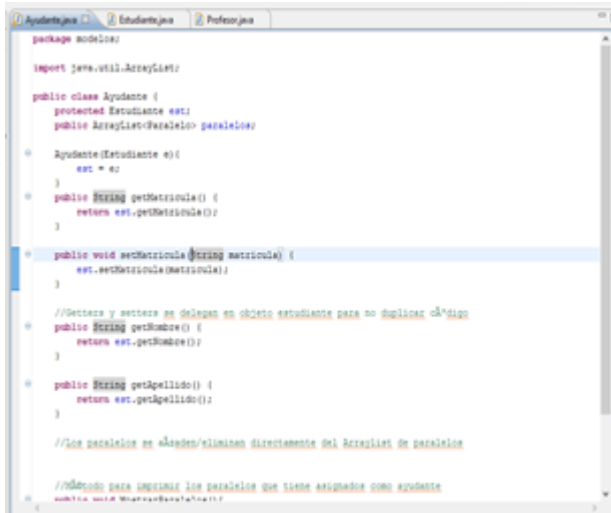
Haciendo un análisis el parámetro `Paralelo` puede ser extraído dado que este parámetro solo sirve para verificar si el paralelo existe en la lista del estudiante, esto no tendría mucho sentido, porque no se hace modificación alguna con respecto a algún campo del estudiante, por lo que esta lógica puede ser eliminada. Además, los otros parámetros como `nexámen`, `ndeberes`, `nlecciones` y `ntalleres` pueden ser reemplazados por un objeto, para esto se crea una clase que encapsula estos parámetros llamada `Libreta`, solucionando así el code smell con el método "Introduce Parameter Object".

## Código modificado

```
1 package modelos;
2
3 public class CalculodeNotas {
4
5
6     //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
7     public static double CalcularNotaInicial(Libreta t){
8
9         double notaInicial=0;
10         double notaTeorico=(t.getHexamen()+t.getDeberes()+t.getLecciones())*0.80;
11         double notaPractico=(t.getTalleres())*0.20;
12         notaInicial=notaTeorico+notaPractico;
13
14         return notaInicial;
15     }
16
17     //Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
18
19     public static double CalcularNotaFinal(Libreta t){
20
21         double notaFinal=0;
22         double notaTeorico=(t.getHexamen()+t.getDeberes()+t.getLecciones())*0.80;
23         double notaPractico=(t.getTalleres())*0.20;
24         notaFinal=notaTeorico+notaPractico;
25
26         return notaFinal;
27     }
28
29
30     //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. Esta nota es solo el promedio de las dos calificaciones anterior
31     public static double CalcularNotaTotal(Paralelo p){
32         double notaTotal=0;
33
34         notaTotal=(p.getMateria().notaInicial+p.getMateria().notaFinal)/2;
35
36         return notaTotal;
37     }
38 }
39
40
```

# Code Smell: Feature Envy

Código inicial / code smell



```
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();
    }

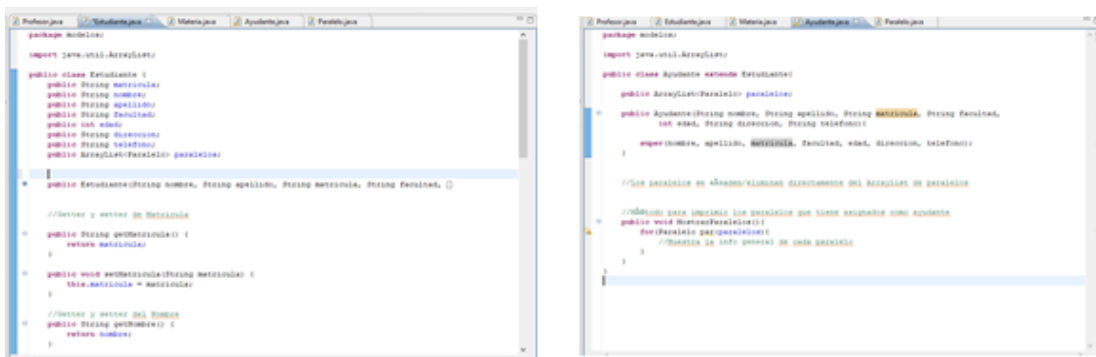
    public String getApellido() {
        return est.getApellido();
    }

    //Los paralelos se añaden/eliminan directamente del ArrayList de paralelos
    //Método para imprimir los paralelos que tiene asignados como ayudante
    public void mostrarParalelos() {
    }
```

## Consecuencias y técnicas de refactorización

En este código se identificó el code Smell Feature Envy ya que la clase Ayudante usa de manera excesiva los métodos de la clase Estudiante. Cada uno de los métodos implementados en esta clase delegan el trabajo a los métodos de Estudiante. En vista de que estas clases comparten atributos y comportamientos similares, se puede resolver este codeSmell haciendo uso del polimorfismo para evitar esta delegación de comportamiento y más bien heredarlos de una clase padre, esta técnica de refactorización es conocida como Replace Delegation with Inheritance.

## Código modificado



```
package modelos;

import java.util.ArrayList;

public class Estudiante {
    public String matricula;
    public String nombre;
    public String apellido;
    public String direccion;
    public int edad;
    public String telefono;
    public ArrayList<Paralelo> paralelos;

    public Estudiante(String nombre, String apellido, String matricula, String direccion, int edad, String telefono, ArrayList<Paralelo> paralelos) {
        //Setters y setters de Estudiante
        this.nombre = nombre;
        this.apellido = apellido;
        this.matricula = matricula;
        this.direccion = direccion;
        this.edad = edad;
        this.telefono = telefono;
        this.paralelos = paralelos;
    }

    //Getters y setters de Estudiante
    public String getMatricula() {
        return matricula;
    }

    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }

    //Getters y setters de Estudiante
    public String getNombre() {
        return nombre;
    }

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

```
package modelos;

import java.util.ArrayList;

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

    public Ayudante(String nombre, String apellido, String matricula, String direccion, int edad, String telefono, ArrayList<Paralelo> paralelos) {
        super(nombre, apellido, matricula, direccion, edad, telefono, paralelos);
    }

    //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 p: paralelos) {
            //Muestra la info general de cada paralelo
        }
    }
}
```

# Code Smell: Data class

## Código inicial / code smell

```
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 }
12
```

## Consecuencias y técnicas de refactorización

En la clase Materia se puede observar que sólo contiene atributos y no funcionalidades extras, lo cual hace que la clase solo sirva como reserva de data que otra clase podría utilizar (como es en el de calcularNota). Aquí no se está aprovechando la usabilidad que tiene una clase como tal, y es por eso que posee el code smell de Data Class. Las consecuencias de esto es que la clase se vuelve muy “estática” y se hace un poco más difícil entender luego el código.

Para corregir este code smell se utilizará la técnica de refactorización Encapsulated Field, el cual a los atributos públicos se los hará privados y además se le añadirá a la clase su respectivo constructor más los getters y setters.



## Código modificado

```
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     public Materia(String codigo, String nombre, String facultad) {
12         super();
13         this.codigo = codigo;
14         this.nombre = nombre;
15         this.facultad = facultad;
16     }
17
18     public String getCodigo() {
19         return codigo;
20     }
21     public void setCodigo(String codigo) {
22         this.codigo = codigo;
23     }
24     public String getNombre() {
25         return nombre;
26     }
27     public void setNombre(String nombre) {
28         this.nombre = nombre;
29     }
30     public String getFacultad() {
31         return facultad;
32     }
33     public void setFacultad(String facultad) {
34         this.facultad = facultad;
35     }
36
37     public double getNotaInicial() {
38         return notaInicial;
39     }
40     public void setNotaInicial(double notaInicial) {
41         this.notaInicial = notaInicial;
42     }
43     public double getNotaFinal() {
44         return notaFinal;
45     }
46     public void setNotaFinal(double notaFinal) {
47         this.notaFinal = notaFinal;
48     }
49     public double getNotaTotal() {
50         return notaTotal;
51     }
52     public void setNotaTotal(double notaTotal) {
53         this.notaTotal = notaTotal;
54     }
55 }
```

# Code Smell: Duplicate code

## Código inicial / code smell

```
3 public class CalculodeNotas {
4
5
6 //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico
7 public static double CalcularNotaInicial(Libreta t){
8
9     double notaInicial=0;
10    double notaTeorico=(t.getNexamen()+t.getNdeberes()+t.getNlecciones())*0.80;
11    double notaPractico=(t.getNtalleres())*0.20;
12    notaInicial=notaTeorico+notaPractico;
13
14    return notaInicial;
15 }
16
17 //Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teorico y el practico
18
19 public static double CalcularNotaFinal(Libreta t){
20
21     double notaFinal=0;
22     double notaTeorico=(t.getNexamen()+t.getNdeberes()+t.getNlecciones())*0.80;
23     double notaPractico=(t.getNtalleres())*0.20;
24     notaFinal=notaTeorico+notaPractico;
25
26
27     return notaFinal;
28 }
29 }
```

## Consecuencias y técnicas de refactorización

Luego de haber trasladado los métodos `CalcularNotaInicial` y `CalcularNotaFinal` de la clase `Estudiante` a la clase `CalculodeNotas`, se puede notar que ambos métodos a pesar de tener nombres diferentes, ambos poseen igual funcionalidad, es decir, poseen el mismo código. A este code smell se le llama código duplicado, y las consecuencias de tenerlo es que hace que la clase sea más extensa, lo que dificulta su legibilidad y que esté propenso a tener más errores y fallas en seguridad.

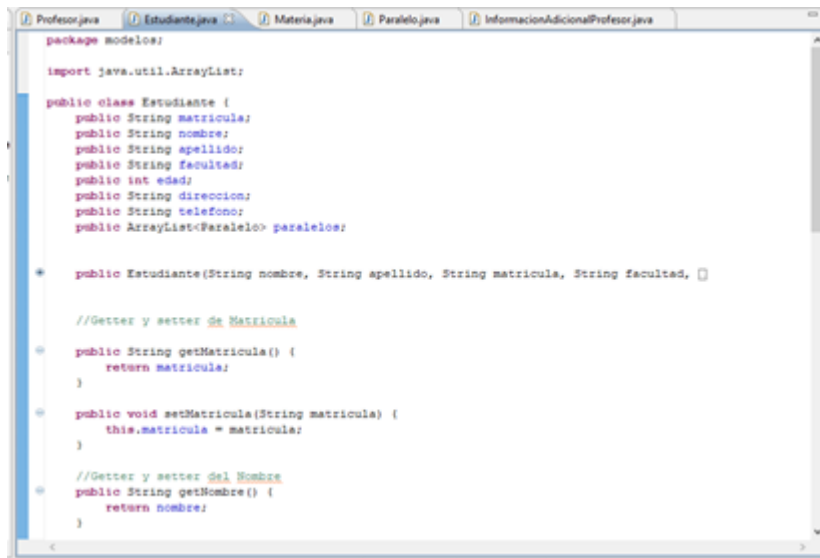
Para corregir este code smell se aplicará la técnica de refactorización `Extract Method`, donde se implementará un nuevo método llamado `CalcularNota` que tenga el mismo código que de ambos métodos duplicados, y posteriormente estos serán suprimidos.

## Código modificado

```
1 package modelos;
2
3 public class CalculodeNotas {
4
5
6     //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico
7     public static double CalcularNota(Libreta t){
8
9         double notaInicial=0;
10        double notaTeorico=(t.getNexamen()+t.getNdeberes()+t.getNlecciones())*0.80;
11        double notaPractico=(t.getNtalleres())*0.20;
12        notaInicial=notaTeorico+notaPractico;
13
14        return notaInicial;
15    }
16
17    //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. Esta nota s
18    public static double CalcularNotaTotal(Paralelo p){
19        double notaTotal=0;
20
21        notaTotal=(p.getMateria().notaInicial+p.getMateria().notaFinal)/2;
22
23        return notaTotal;
24    }
25 }
26
27
```

# Code Smell: Inappropriate intimacy

Código inicial / code smell



```
package modelos;

import java.util.ArrayList;

public class 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;

    * public Estudiante(String nombre, String apellido, String matricula, String facultad, )

    //Getter y setter de Matricula
    public String getMatricula() {
        return matricula;
    }

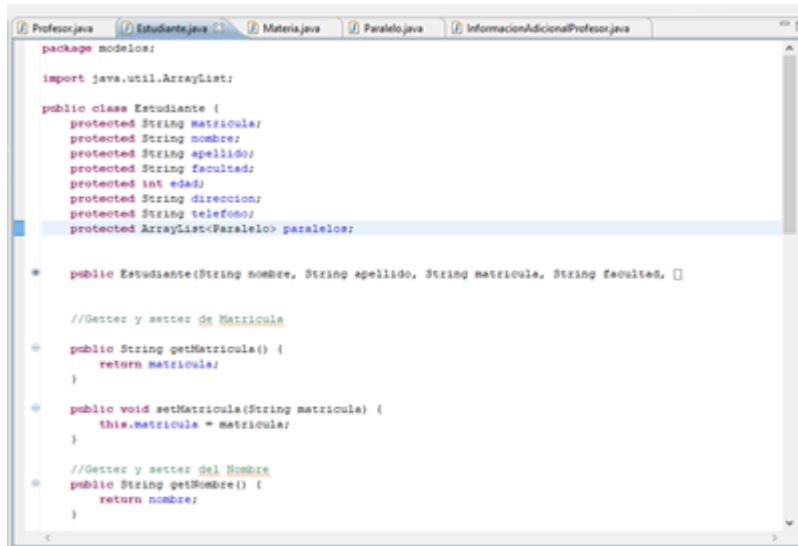
    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }

    //Getter y setter del Nombre
    public String getNombre() {
        return nombre;
    }
}
```

## Consecuencias y técnicas de refactorización

En la clase Estudiante se puede apreciar que todos los atributos contenidos tienen el modificador de acceso público, lo cual permite que sean accedidos desde clases externas, esto es reconocido como el code Smell Inappropriate intimacy, para solucionarlo se cambiarán los modificadores de acceso a `protected` para que las únicas que tengan acceso a esta información sean las clases derivadas, esta encapsulación de atributos es conocida como una técnica de refactorización llamada: Encapsulated field.

## Código modificado



```
package modelos;

import java.util.ArrayList;

public class Estudiante {
    protected String matricula;
    protected String nombre;
    protected String apellido;
    protected String facultad;
    protected int edad;
    protected String direccion;
    protected String telefono;
    protected ArrayList<Paralelo> paralelos;

    * public Estudiante(String nombre, String apellido, String matricula, String facultad, )

    //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;
    }
}
```