

Taller Refactoring

Diseño de Software

Integrantes:

- Hatus Pellegrini
- Joby Farra
- Josue Montoya
- Oscar Lucas

ESPOL 2T-2020



Tabla de contenido

1. Code Smells: Duplicate Code	3
1.1. Consecuencias.....	3
1.2. Técnica de Refactoring	3
1.3. Código Inicial.....	3
1.4. Código Final.....	3
2. Code Smells: Feacture Envy	4
2.1. Consecuencia	4
2.2. Técnica de Refactorización.....	4
2.3. Código Inicial.....	4
2.4. Código Final.....	5
3. Code Smell: Comments	6
3.1 Consecuencia	6
3.2 Técnica de Refactorización.....	6
3.3 Código Inicial.....	6
3.4 Código Final.....	6
4 Code Smell: Long Class.....	7
4.1 Consecuencias.....	7
4.2 Técnica de Refactorización.....	7
4.3 Código Inicial.....	7
4.4 Código Final.....	7
5. Code Smell: Data class	8
5.1 Código initial / code smell	8
5.2 Consecuencias.....	8
5.3 Técnicas de refactorización	8
5.4 Código modificado	9
6. Code Smell: Inappropriate intimacy	10
6.1 Consecuencia	10
6.2 Técnica de Refactorización.....	10
6.3 Código Inicial.....	10
6.4 Código Modificado	10

1. Code Smells: Duplicate Code

1.1. Consecuencias

El code smells encontrado en la siguiente parte de código es duplicate code ya que existe un código fuente que ocurre mas de una vez, dentro de un programa o a través de diferentes programas. La consecuencia de esto radica en el espacio de memoria, el código duplicado ocupa memoria innecesaria lo cual se realizan procesos que hacen el código mas pesado y hace al programador confundirse.

1.2. Técnica de Refactoring

Extract Method es una técnica de Refactoring que nos ayuda cuando mas líneas se encuentren en un método, mas difícil averiguar qué hace el método. Entonces agregamos en una función todo ese código que cuando se necesita se llame.

1.3. Código Inicial

```
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;
}
```

1.4. Código Final

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

2. Code Smells: Feature Envy

2.1. Consecuencia

El code smells encontrado en el siguiente bloque de código es Feature Envy el cual lo identificamos cuando un objeto está mas interesado en otro que en su mismo. Existe mucho acoplamiento entre la clase ayudante y estudiante, la consecuencia con esto es que si se elimina una parte de código en la clase Estudiante puede afectar al funcionamiento de la clase Ayudante.

2.2. Técnica de Refactorización

Reemplazar Delegación por Herencia es una técnica de refactoring dice que podemos usar la herencia para para lograr que la clase estudiante y la clase ayudante trabajen en códigos separados para que no haya necesidad de que haya código duplicado. Además, se logra que la clase ayudante se construya con menos código y sea posible mantenerlo a lo largo del tiempo.

2.3. Código Inicial

```
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 estu
    public String getNombre() {
        return est.getNombre();
    }

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

2.4. Código Final

```
package modelos;

import java.util.ArrayList;

public class Ayudante extends Estudiante {

    private ArrayList<Paralelo> paralelos;

    public ArrayList<Paralelo> paralelos(){
        return paralelos;
    }

    public void setParalelos(ArrayList<Paralelo> paralelos){
        this.paralelos = this.paralelos;
    }

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

3. Code Smell: Comments

3.1 Consecuencia

El code smell se evidencia en la clase Ayudante, donde para el metodo MostrarParalelos () se explica con 3 comments su funcionamiento. Esto no debería ser así, dado que el código debe desarrollarse de manera que pueda ser entendido por cualquiera que lo vea sin necesidad de comentarios adicionales.

3.2 Técnica de Refactorización

Para solucionarlo, se utiliza la técnica de Rename Method, en donde se cambia el nombre del metodo por uno más intuitivo sobre su funcionamiento, para que los comentarios no sean necesarios para su comprensión.

3.3 Código Inicial

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

3.4 Código Final

```
public void MostrarInfoParalelos(){
    for(Paralelo par:paralelos){
        //Muestra la info general de cada paralelo
    }
}
```

4 Code Smell: Long Class

4.1 Consecuencias

Este code smell se puede identificar en la clase Estudiante, la cual es sumamente extensa y puede llegar a causar confusión al momento de leer el código o de tratar de solucionar algún bug.

4.2 Técnica de Refactorización

Para solucionar este code Smell se utiliza la técnica de refactorización de Extract Class, en donde parte del código dentro de la clase Estudiante se separa a otro componente, en este caso una clase Calculadora Denotas, la cual se encarga únicamente de calcular la nota dado un estudiante.

4.3 Código Inicial

```
//Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
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 por parcial.
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;
}

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

4.4 Código Final

```
public class CalculadoraDeNotas {

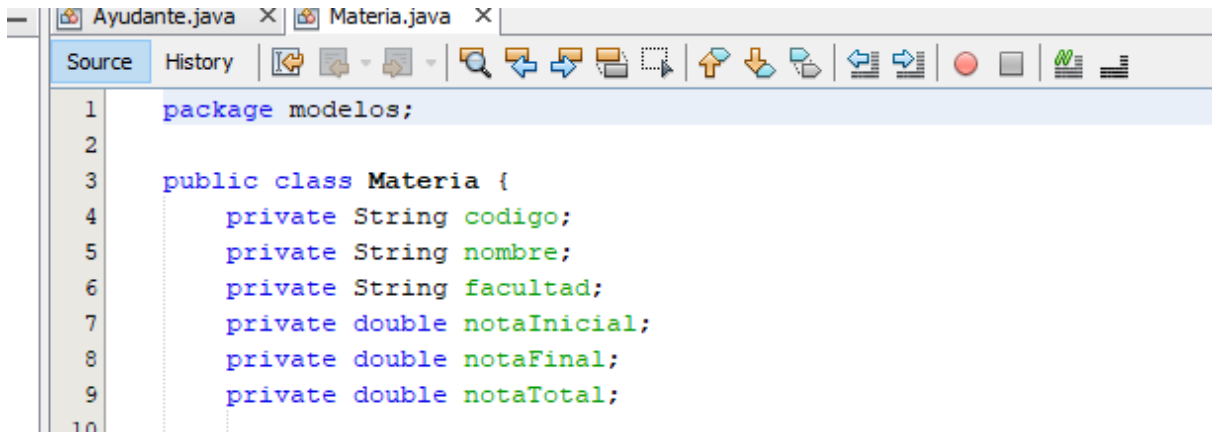
    //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
    public double CalcularNotaInicial(Estududiante e, Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres) {
        double notaInicial = 0;
        for (Paralelo par : e.paralelos) {
            if (p.equals(par)) {
                double notaTeorico = (e.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 por parcial.
    public double CalcularNotaFinal(Estududiante e, Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres) {
        double notaFinal = 0;
        for (Paralelo par : e.paralelos) {
            if (p.equals(par)) {
                double notaTeorico = (nexamen + ndeberes + nlecciones) * 0.80;
                double notaPractico = (ntalleres) * 0.20;
                notaFinal = notaTeorico + notaPractico;
            }
        }
        return notaFinal;
    }

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

5. Code Smell: Data class

5.1 Código inicial / code smell



```
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 }
```

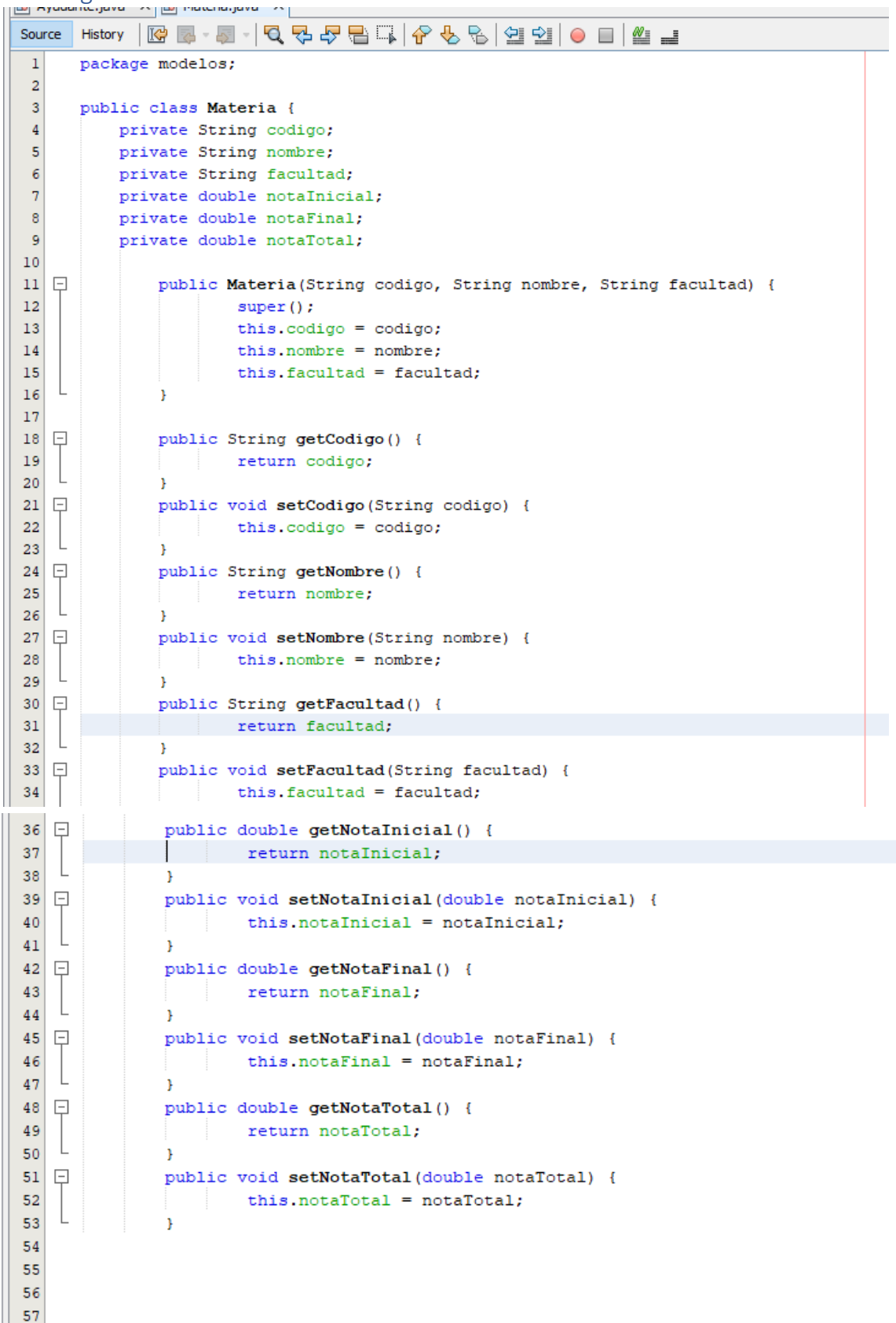
5.2 Consecuencias

En la clase `Materia` se puede evidenciar 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. 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.

5.3 Técnicas de refactorización

Como solución para 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.

5.4 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     public double getNotaInicial() {
37         return notaInicial;
38     }
39     public void setNotaInicial(double notaInicial) {
40         this.notaInicial = notaInicial;
41     }
42     public double getNotaFinal() {
43         return notaFinal;
44     }
45     public void setNotaFinal(double notaFinal) {
46         this.notaFinal = notaFinal;
47     }
48     public double getNotaTotal() {
49         return notaTotal;
50     }
51     public void setNotaTotal(double notaTotal) {
52         this.notaTotal = notaTotal;
53     }
54
55
56
57
```

6. Code Smell: Inappropriate intimacy

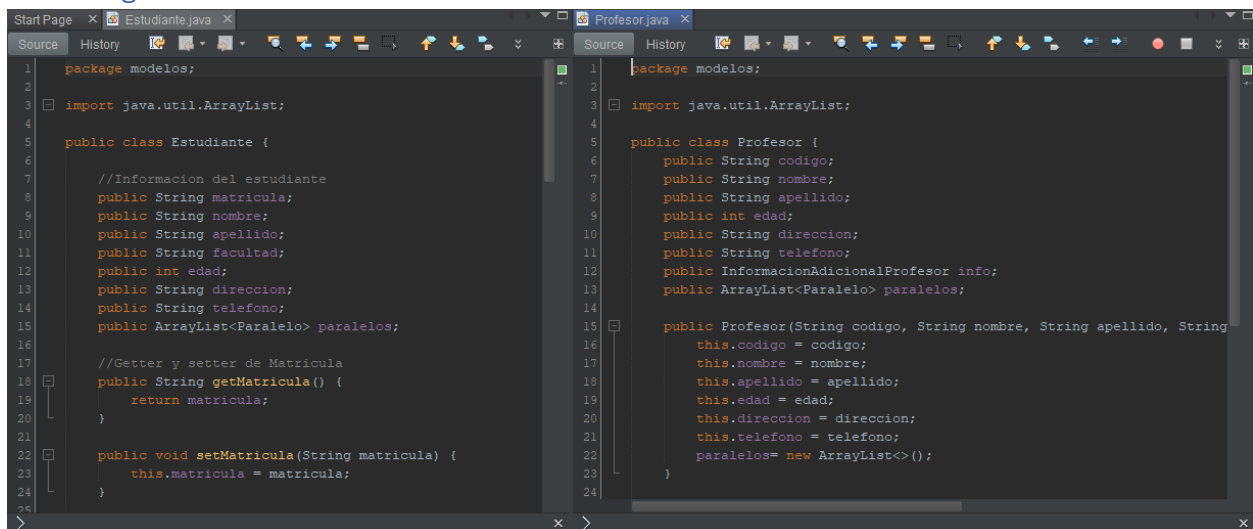
6.1 Consecuencia

En la clase Estudiante y Profesor, se puede apreciar que los atributos que poseen estas clases tienen un modificador de acceso “public”, por lo que permite que estos atributos sean accedidos desde clases externas. Por lo que este code smell se lo conoce como Inappropriate intimacy.

6.2 Técnica de Refactorización

Para solucionar este code smell encontrado en estas clases, se puede aplicar la técnica de refactorización llamada “Encapsulated Field”, la cual nos permite cambiar los modificadores de accesos de los atributos a “private” y así permitir el acceso solo a clases derivadas, además permitir el uso por medio de los get and setters.

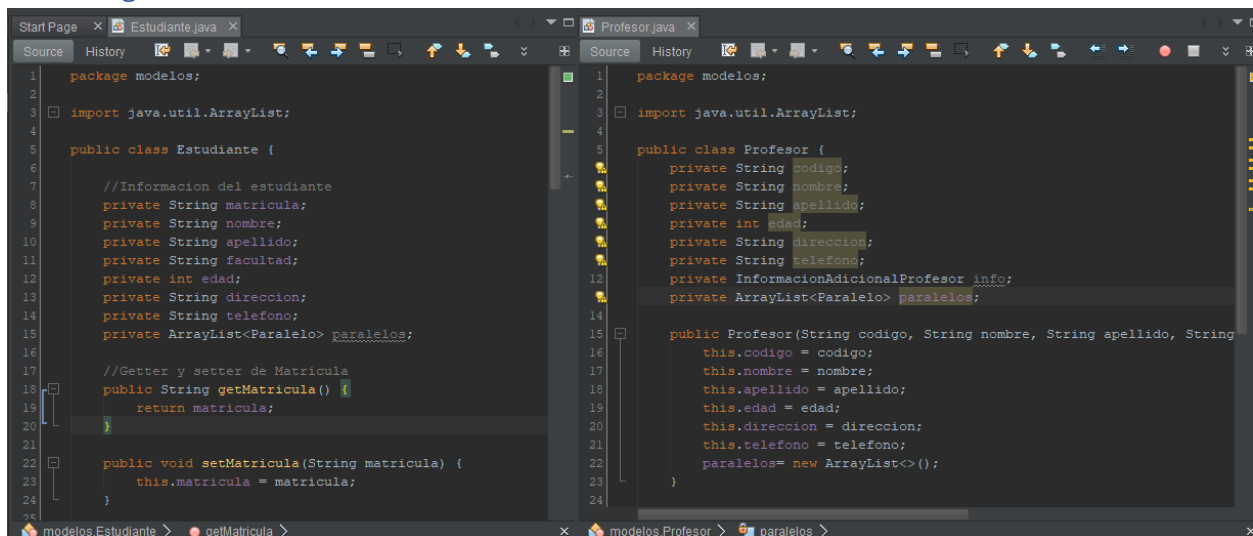
6.3 Código Inicial



```
1 package modelos;
2
3 import java.util.ArrayList;
4
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    public String getMatricula() {
18        return matricula;
19    }
20
21    public void setMatricula(String matricula) {
22        this.matricula = matricula;
23    }
24
25 }
26
```

```
1 package modelos;
2
3 import java.util.ArrayList;
4
5 public class Profesor {
6     public String codigo;
7     public String nombre;
8     public String apellido;
9     public int edad;
10    public String direccion;
11    public String telefono;
12    public InformacionAdicionalProfesor info;
13    public ArrayList<Paralelo> paralelos;
14
15    public Profesor(String codigo, String nombre, String apellido, String
16        this.codigo = codigo;
17        this.nombre = nombre;
18        this.apellido = apellido;
19        this.edad = edad;
20        this.direccion = direccion;
21        this.telefono = telefono;
22        paralelos = new ArrayList<>();
23    }
24 }
```

6.4 Código Modificado



```
1 package modelos;
2
3 import java.util.ArrayList;
4
5 public class Estudiante {
6     //Informacion del estudiante
7     private String matricula;
8     private String nombre;
9     private String apellido;
10    private String facultad;
11    private int edad;
12    private String direccion;
13    private String telefono;
14    private ArrayList<Paralelo> paralelos;
15
16    //Getter y setter de Matricula
17    public String getMatricula() {
18        return matricula;
19    }
20
21    public void setMatricula(String matricula) {
22        this.matricula = matricula;
23    }
24
25 }
26
```

```
1 package modelos;
2
3 import java.util.ArrayList;
4
5 public class Profesor {
6     private String codigo;
7     private String nombre;
8     private String apellido;
9     private int edad;
10    private String direccion;
11    private String telefono;
12    private InformacionAdicionalProfesor info;
13    private ArrayList<Paralelo> paralelos;
14
15    public Profesor(String codigo, String nombre, String apellido, String
16        this.codigo = codigo;
17        this.nombre = nombre;
18        this.apellido = apellido;
19        this.edad = edad;
20        this.direccion = direccion;
21        this.telefono = telefono;
22        paralelos = new ArrayList<>();
23    }
24 }
```