

TALLER REFACTORING

DISEÑO DE SOFTWARE

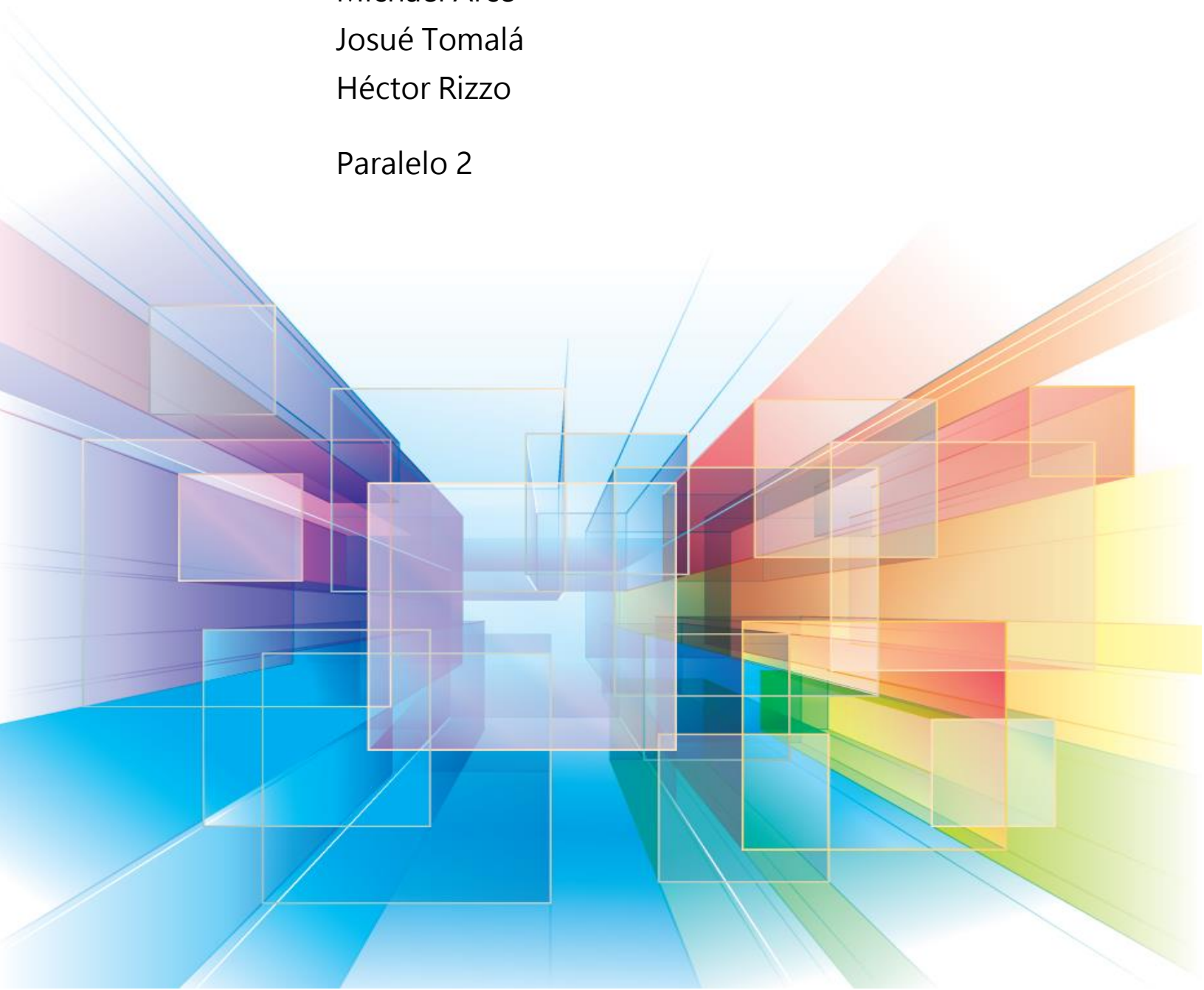
INTEGRANTES

Michael Arce

Josué Tomalá

Héctor Rizzo

Paralelo 2



Contenido

Feature Envy	3
Consecuencias.....	3
Remplazar Delegación con Herencia	3
Duplicated Code.....	4
Consecuencias.....	4
Extract Method	4
Speculative Generality	5
Consecuencias.....	6
Remove setting Methods.....	6
Duplicate Code y Shotgun surgery.....	9
Consecuencias.....	9
Extract Class	9
Inappropriate Intimacy.....	10
Consecuencias.....	10
Solución.....	11
Long Parameter List	12
Consecuencias.....	13
Solución.....	13
Data Class.....	13
Consecuencias.....	14
Solución.....	14
Lazy Class.....	14
Consecuencias.....	14
Solución.....	15

Repositorio

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

Feature Envy

La clase Ayudante recibe un objeto de clase Estudiante y todos sus métodos acceden a los campos del estudiante.

Consecuencias

Aparte de tener código repetido, estamos accediendo a todos los campos de estudiantes por lo que tranquilamente se los podría heredar. Además que si se hace un cambio en estudiante, esto afectaría y debería ser corregido en esta clase.

Remplazar Delegación con Herencia

Lo que se realizara es que el ayudante sea hijo de la clase estudiante, ya que contiene todos sus campos, y con esto los cambios también se compartirán entre clases.

Problema

```
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();
    }
}
```

Solución

```
public class Ayudante extends Estudiante{
    public ArrayList<Paralelo> ayudantias;
    Ayudante(){
        super();
    }

    public ArrayList<Paralelo> getAyudantias() {
        return ayudantias;
    }

    public void setAyudantias(ArrayList<Paralelo> ayudantias) {
        this.ayudantias = ayudantias;
    }

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

Duplicated Code

Código duplicado que hace más larga la clase y nos brinda más líneas para leer.

Consecuencias

Tener código duplicado nos incrementa el tiempo de lectura, además de aquello no es eficiente porque si queremos hacer un cambio, tendríamos que buscar en todas las líneas donde se encuentra el código. Por ejemplo, en el proyecto se calcula la nota de un paralelo con un valor fijo que se multiplica a las notas de los componentes, entonces si este valor lo decidiéramos cambiar, tendríamos que buscar en todo el proyecto en donde se repiten esta línea.

Extract Method

Usar Extract Method, lo cual hace que tengamos solo un bloque con esta declaración de código y lo usemos en los otros métodos que lo necesite.

Problema

```

81 //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teórico 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 }

```

```

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;

```

Solución

```

81 //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teórico y el practico se calcula por parcial.
82 public double CalcularNotaInicial(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres) {
83     return calcularNota(p, nexamen, ndeberes, nlecciones, ntalleres);
84 }
85
86 //Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teórico y el practico se calcula por parcial.
87 public double CalcularNotaFinal(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres) {
88     return calcularNota(p, nexamen, ndeberes, nlecciones, ntalleres);
89 }
90
91 private double calcularNota(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres) {
92     double notaFinal = 0;
93     for (Paralelo par : paralelos) {
94         if (p.equals(par)) {
95             double notaTeorico = (nexamen + ndeberes + nlecciones) * 0.80;
96             double notaPractico = (ntalleres) * 0.20;
97             notaFinal = notaTeorico + notaPractico;
98         }
99     }
100     return notaFinal;
101 }

```

Speculative Generality

Existen métodos que no son usados y han sido escritos para un uso a futuro como algunos getters y setters en las clases.

Consecuencias

Las clases son más largas y es más difícil de entender el código y de darle soporte.

Remove setting Methods

Se remueven los métodos de setters y otros métodos que no son usados lo cual nos permite tener un código más limpio más legible y que podamos entender mejor.

Problema

Clase Estudiante:

```
40 public void setApellido(String apellido) { this.apellido = apellido; }
43
44 //Getter y setter de la Facultad
45 public String getFacultad() { return facultad; }
48
49 public void setFacultad(String facultad) { this.facultad = facultad; }
52
53 //Getter y setter de la edad
54 public int getEdad() { return edad; }
57
58 public void setEdad(int edad) { this.edad = edad; }
61
62 //Getter y setter de la direccion
63 public String getDireccion() { return direccion; }
66
67 public void setDireccion(String direccion) { this.direccion = direccion; }
70
71 //Getter y setter del telefono
72
73 public String getTelefono() { return telefono; }
76
77 public void setTelefono(String telefono) { this.telefono = telefono; }
80
```

Clase Paralelo:

```
12 public int getNumero() { return numero; }
15
16 public void setNumero(int numero) { this.numero = numero; }
19
20 public Materia getMateria() { return materia; }
23
24 public void setMateria(Materia materia) { this.materia = materia; }
27
28 public Profesor getProfesor() { return profesor; }
31
32 public void setProfesor(Profesor profesor) { this.profesor = profesor; }
35
36 //Imprime el listado de estudiantes registrados
37 public void mostrarListado() {
38     //No es necesario implementar
39 }
40
```

Solución

```
public class Paralelo {  
    public int numero;  
    public Materia materia;  
    public Profesor profesor;  
    public ArrayList<Estudiante> estudiantes;  
    public Ayudante ayudante;  
  
    public Materia getMateria() {  
        return materia;  
    }  
  
    //Imprime el listado de estudiantes registrados  
    public void mostrarListado(){  
        //No es necesario implementar  
    }  
}
```

```
public class Persona {  
    //Informacion de la persona  
    public String codigo;  
    public String nombre;  
    public String apellido;  
    public String facultad;  
    public int edad;  
    public String direccion;  
    public String telefono;  
    public ArrayList<Paralelo> paralelos;  
  
    public Persona(String codigo, String nombre, String apellido, int edad) {  
        this.codigo = codigo;  
        this.nombre = nombre;  
        this.apellido = apellido;  
        this.edad = edad;  
    }  
  
    public ArrayList<Paralelo> getParalelos() { return paralelos; }  
}
```


Duplicate Code y Shotgun surgery

Podemos notar que existe código duplicado de información, tanto en profesor como en estudiante.

Consecuencias

La mala implementación de estos fields nos podría producir shotgun surgery, ya que en el caso de que se quiera agregar o quitar información de una persona independientemente si sea profesor o estudiante, debemos de modificar en las dos clases.

Extract Class

Con esto crearemos una nueva clase llamada Persona, la cual será el padre de Estudiante y profesor para no tener este código duplicado.

Problema

```
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, S
        this.codigo = codigo;
        this.nombre = nombre;
        this.apellido = apellido;
        this.edad = edad;
        this.direccion = direccion;
        this.telefono = telefono;
        paralelos= new ArrayList<>();
    }
}

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;
    }
}
```

Solución

```
public class Persona {  
    //Informacion de la persona  
    public String codigo;  
    public String nombre;  
    public String apellido;  
    public String facultad;  
    public int edad;  
    public String direccion;  
    public String telefono;  
    public ArrayList<Paralelo> paralelos;  
  
    public Persona(String codigo, String nombre, String apellido, int edad) {  
        this.codigo = codigo;  
        this.nombre = nombre;  
        this.apellido = apellido;  
        this.edad = edad;  
    }  
}
```

Inappropriate Intimacy

Este code smell lo podemos encontrar en todas las clases del proyecto, ya que todas poseen campos públicos.

Consecuencias

Podrán existir clases que utilicen estos campos que no les pertenecen, provocando un mayor acoplamiento entre las clases.

Problema

```

or.java  Persona.java  Paralelo.java*  Materia.java  In
1  package modelos;
2
3  import java.util.ArrayList;
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 Materia getMateria() {
13         return materia;
14     }
15
16     //Imprime el listado de estudiantes registrados
17     public void mostrarListado(){
18         //No es necesario implementar
19     }
20 }

```

```

or.java  Persona.java  Paralelo.java*  Materia.java  X
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

```

Solución

Técnica Encapsulate Field: Tendremos que colocar modificadores de acceso privados o protected (dependiendo de los requerimientos) y utilizar métodos *Getters and Setters* para poder acceder a estos campos. (En las capturas solo se muestran dos clases, pero los cambios se realizaron en todas las clases)

Solución

```

esor.java*  Persona.java*  Paralelo.java*  X  Materia.java*
1  package modelos;
2
3  import java.util.ArrayList;
4
5  public class Paralelo {
6      private int numero;
7      private Materia materia;
8      private Profesor profesor;
9      private ArrayList<Estudiante> estudiantes;
10     private Ayudante ayudante;
11
12     public Materia getMateria() {
13         return materia;
14     }
15
16     //Imprime el listado de estudiantes registrados
17     public void mostrarListado(){
18         //No es necesario implementar
19     }
20
21     //Métodos Getters and Setters
22 }

```

```

esor.java*  Materia.java*  X  Persona.java*  Paralelo.java*
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 c, String n, String f) {
12         codigo = c;
13         nombre = n;
14         facultad = f;
15     }
16
17     //Métodos Getters and Setters
18 }

```

Long Parameter List

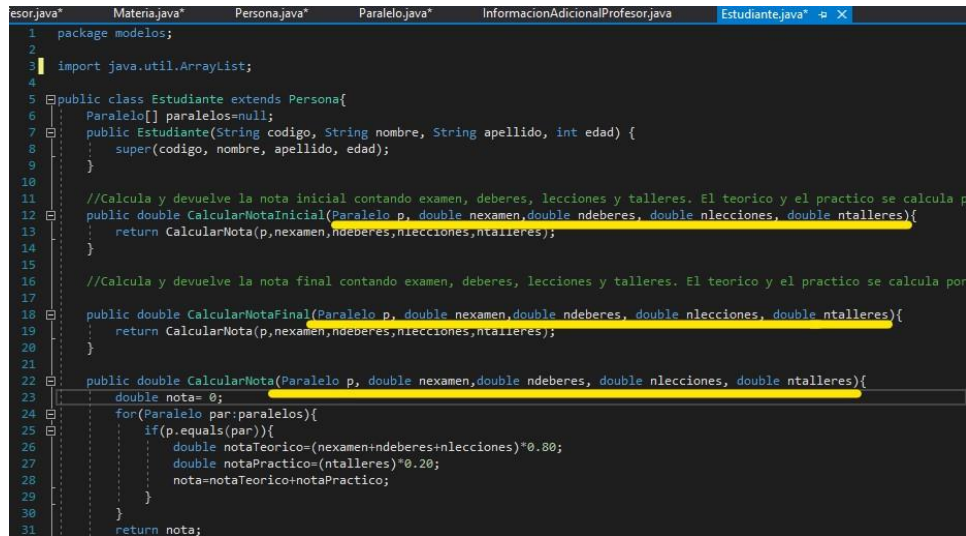
Los siguientes métodos contienen este code smell:

- Clase Estudiante:
 - Método CalcularNotaInicial: Contiene un total 5 parámetros.
 - Método CalcularNotaFinal: Contiene un total de 5 parámetros.

Consecuencias

Esto provoca una peor legibilidad del código y reduciría su comprensión.

Problema



```

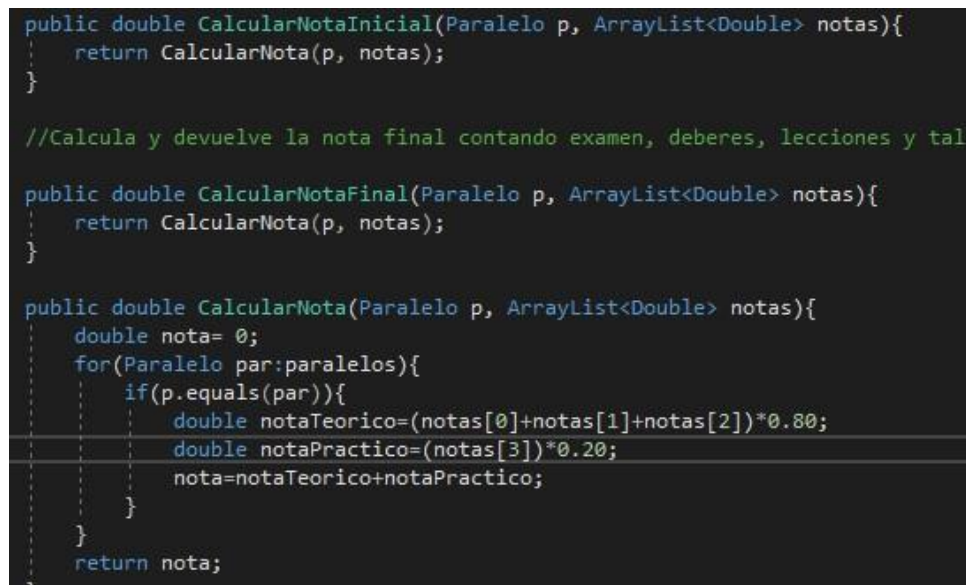
1 package modelos;
2
3 import java.util.ArrayList;
4
5 public class Estudiante extends Persona{
6     Paralelo[] paralelos=null;
7     public Estudiante(String codigo, String nombre, String apellido, int edad) {
8         super(codigo, nombre, apellido, edad);
9     }
10
11     //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula p
12     public double CalcularNotaInicial(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres){
13         return CalcularNota(p, nexamen, ndeberes, nlecciones, ntalleres);
14     }
15
16     //Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por
17
18     public double CalcularNotaFinal(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres){
19         return CalcularNota(p, nexamen, ndeberes, nlecciones, ntalleres);
20     }
21
22     public double CalcularNota(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres){
23         double nota= 0;
24         for(Paralelo par:paralelos){
25             if(p.equals(par)){
26                 double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
27                 double notaPractico=(ntalleres)*0.20;
28                 nota=notaTeorico+notaPractico;
29             }
30         }
31         return nota;

```

Solución

Técnica Introduce Parameter Object: Para reducir la cantidad, utilizaremos esta técnica ya que tenemos una serie de parámetros del mismo tipo. Para reducir esto, reemplazaremos estos con un arreglo, así reduciríamos la cantidad a 2.

Solución



```

public double CalcularNotaInicial(Paralelo p, ArrayList<Double> notas){
    return CalcularNota(p, notas);
}

//Calcula y devuelve la nota final contando examen, deberes, lecciones y tal

public double CalcularNotaFinal(Paralelo p, ArrayList<Double> notas){
    return CalcularNota(p, notas);
}

public double CalcularNota(Paralelo p, ArrayList<Double> notas){
    double nota= 0;
    for(Paralelo par:paralelos){
        if(p.equals(par)){
            double notaTeorico=(notas[0]+notas[1]+notas[2])*0.80;
            double notaPractico=(notas[3])*0.20;
            nota=notaTeorico+notaPractico;
        }
    }
    return nota;
}

```

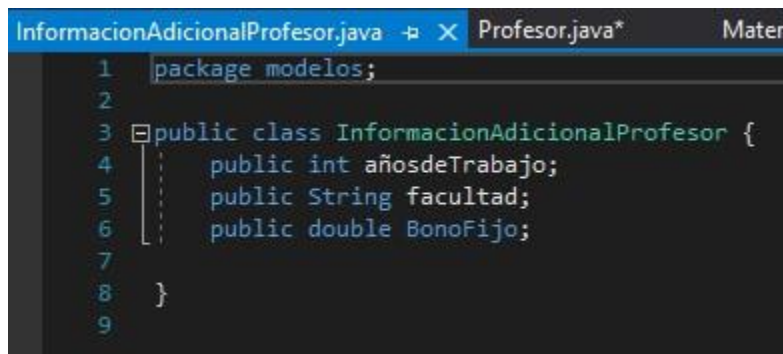
Data Class

La clase 'InformacionAdicionalProfesor' solo contiene campos públicos sin ningún método.

Consecuencias

Es una clase que no tiene ninguna funcionabilidad, solo es complementarias para la clase Profesor.

Problema

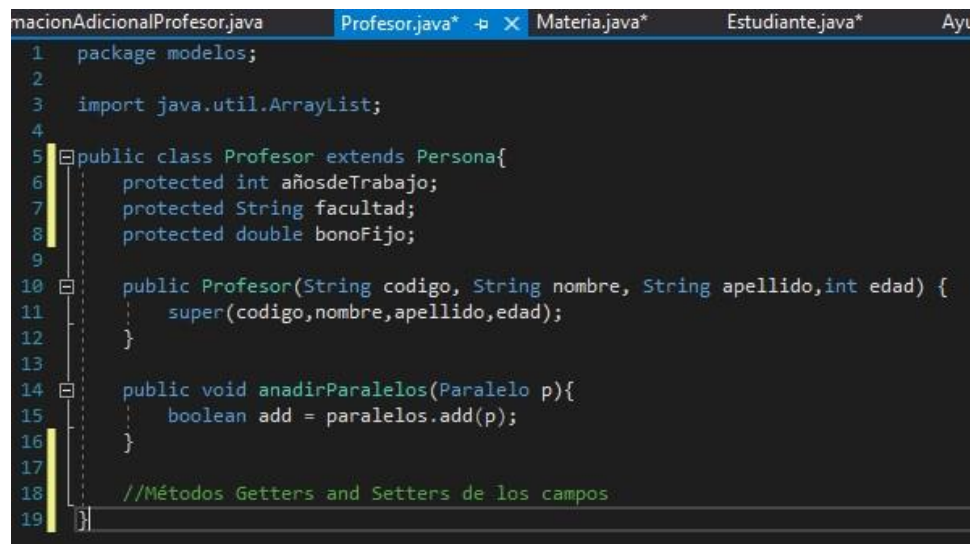


```
1 package modelos;
2
3 public class InformacionAdicionalProfesor {
4     public int añosdeTrabajo;
5     public String facultad;
6     public double BonoFijo;
7
8 }
9
```

Solución

Técnica Move Field: Utilizaremos esta técnica ya que moveremos los campos de la clase InformacionAdicionalProfesor a la clase Profesor.

Solución



```
1 package modelos;
2
3 import java.util.ArrayList;
4
5 public class Profesor extends Persona{
6     protected int añosdeTrabajo;
7     protected String facultad;
8     protected double bonoFijo;
9
10    public Profesor(String codigo, String nombre, String apellido,int edad) {
11        super(codigo,nombre,apellido,edad);
12    }
13
14    public void anadirParalelos(Paralelo p){
15        boolean add = paralelos.add(p);
16    }
17
18    //Métodos Getters and Setters de los campos
19 }
```

Lazy Class

La clase calcularSueldoProfesor solo contiene un solo método que se encarga de calcular el sueldo de un objeto de tipo Profesor.

Consecuencias

Debido al pequeño aporte que genera esta clase, nos conviene remover y colocar ese método en otra clase, ya que nos ahorraríamos el mantenimiento que deba tener una clase nueva.

Problema

```

1 package modelos;
2
3 public class calcularSueldoProfesor {
4
5     public double calcularSueldo(Profesor prof){
6         double sueldo=0;
7         sueldo= prof.info.añosdeTrabajo*600 + prof.info.BonoFijo;
8         return sueldo;
9     }
10 }

```

Solución

Técnica Inline Class: Ya que esta clase no presenta muchas funcionalidades, e incluso la clase utiliza los datos del Profesor que recibe, podríamos mover el método a la clase Profesor y remover la clase.

Solución

```

3 import java.util.ArrayList;
4
5 public class Profesor extends Persona{
6     protected int añosdeTrabajo;
7     protected String facultad;
8     protected double bonoFijo;
9
10    public Profesor(String codigo, String nombre, String apellido,int edad) {
11        super(codigo,nombre,apellido,edad);
12    }
13
14    public void anadirParalelos(Paralelo p){
15        boolean add = paralelos.add(p);
16    }
17
18    public double calcularSueldo(){
19        return añosdeTrabajo*600 + bonoFijo;
20    }
21
22    //Métodos Getters and Setters de los campos
23 }

```