



Reporte Factoring

Integrantes:

- Andrea Soriano
- Bryan Puchaicela
- Kenny Yépez

ÍNDICE:

1 [Code Smells](#)

1.1 [Large Class](#)

1.2 [Long Parameter List](#)

1.3 [Data Clumps](#)

1.4 [Temporary Field](#)

1.5 [Alternative Classes with Different Interfaces](#)

1.6 [Data Class](#)

1.7 [Feature Envy](#)

1.8 [Lazy Class](#)

1.9 [Inappropriate Intimacy](#)

1.10 [Duplicated Code](#)

Code Smells

Large Class

Código Original:

```
public class Estudiante{
    //Informacion del estudiante
    public String matricula;
    public String nombres;
    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 nombres;
    }

    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;
    }

    //Getter y setter de la Facultad
    public String getFacultad() {
        return facultad;
    }

    public void setFacultad(String facultad) {
        this.facultad = facultad;
    }

    //Getter y setter de la edad
    public int getEdad() {
        return edad;
    }

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

    //Getter y setter de la direccion
    public String getDireccion() {
        return direccion;
    }

    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }
}

//Getter y setter del telefono
public String getTelefono() {
    return telefono;
}

public void setTelefono(String telefono) {
    this.telefono = telefono;
}

//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;
        }
    }
    return notaTotal;
}
```

Código modificado:

```

package modelos;
import java.util.ArrayList;
public class Materia {
    public String codigo;
    public String nombre;
    public String facultad;
    public double notaInicial;
    public double notaFinal;
    public double notaTotal;
    public ArrayList<Paralelo> paralelos;
    //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;
            }
        }
        return notaTotal;
    }
}

package modelos;
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;

    //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;
    }

    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    //Getter y setter de la Facultad
    public String getFacultad() {
        return facultad;
    }

    public void setFacultad(String facultad) {
        this.facultad = facultad;
    }

    //Getter y setter de la edad
    public int getEdad() {
        return edad;
    }

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

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

    //Getter y setter de la direccion
    public String getDireccion() {
        return direccion;
    }

    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }

    //Getter y setter del telefono
    public String getTelefono() {
        return telefono;
    }

    public void setTelefono(String telefono) {
        this.telefono = telefono;
    }
}

```

Consecuencias:

Dejando la clase igual, haría que los desarrolladores tengan que recordar demasiados atributos y métodos, además, es más probable que exista duplicación de código y funcionalidad.

Técnicas de refactorización: Extract Class

Long Parameter List

Código original:

```
//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;
}
```

Código modificado:

```
//Calcula y devuelve la nota teorica
public double notaTeorico(double nexamen, double ndeberes, double nlecciones) {
    return (nexamen + ndeberes + nlecciones) * 0.80;
}

//Devuelve true o false si esta dentro de ese paralelo
public boolean isParalelo(Paralelo p) {
    for (Paralelo par : paralelos) {
        if (p.equals(par)) {
            return true;
        }
    }
    return false;
}

//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 notaTeorico, double ntalleres) {
    double notaInicial = 0;
    if (isParalelo(p)) {
        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 notaTeorico, double ntalleres) {
    double notaFinal = 0;
    if (isParalelo(p)) {
        double notaPractico = (ntalleres) * 0.20;
        notaFinal = notaTeorico + notaPractico;
    }
    return notaFinal;
}
```

Consecuencias:

Manteniendo el código así será más difícil de leer y más largo, además existiría posibilidad de que haya código duplicado.

Técnicas de refactorización: Replace Parameter with Method Call

Data clumps

Código original:

```

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 }

```

```

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 }

```

Código modificado:

```

public class Persona {

    public String nombre;
    public String apellido;
    public int edad;
    public String direccion;
    public String telefono;
//getters y setters
    public String getNombre() {
        return nombre;
    }

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

    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;
    }

    public String getDireccion() {
        return direccion;
    }

    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }

    public String getTelefono() {
        return telefono;
    }

    public void setTelefono(String telefono) {
        this.telefono = telefono;
    }

}

```

Consecuencias:

Mantener el código tal y como esta haría que este sea menos organizado pues las operaciones sobre datos particulares estarán distribuidas por toda la clase, además mantendría la clase extensa y difícil de comprender.

Técnicas de refactorización: Extract SuperClass

Temporary Field

Código original:

```
//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;
        }
    }
    return notaTotal;
}
}
```

Código modificado:

```
package modelos;
import java.util.ArrayList;
public class Materia {
    public String codigo;
    public String nombre;
    public String facultad;
    public double notaInicial;
    public double notaFinal;
    public double notaTotal;
    public ArrayList<Paralelo> paralelos;
    //Calcula y devuelve la nota teorica
    public double notaTeorico(double nexamen, double ndeberes, double nlecciones) {
        return (nexamen + ndeberes + nlecciones) * 0.80;
    }
    //Devuelve true o false si esta dentro de ese paralelo
    public boolean isParalelo(Paralelo p) {
        for (Paralelo par : paralelos) {
            if (p.equals(par)) {
                return true;
            }
        }
        return false;
    }
    //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 notaTeorico, double ntalleres) {
        if (isParalelo(p)) {
            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 notaTeorico, double ntalleres) {
        if (isParalelo(p)) {
            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) {
        if (isParalelo(p)) {
            notaTotal = (p.getMateria().notaInicial + p.getMateria().notaFinal) / 2;
        }
        return notaTotal;
    }
}
```

Consecuencias:

Mantener el código así, existiría un pequeño desperdicio de memoria al crear variables que no son necesarias, además añadiría líneas innecesarias al código, haciéndolo más extensos.

Técnicas de refactorización: Introduce Null Object

Alternative Classes with Different Interfaces

Código original:

```
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

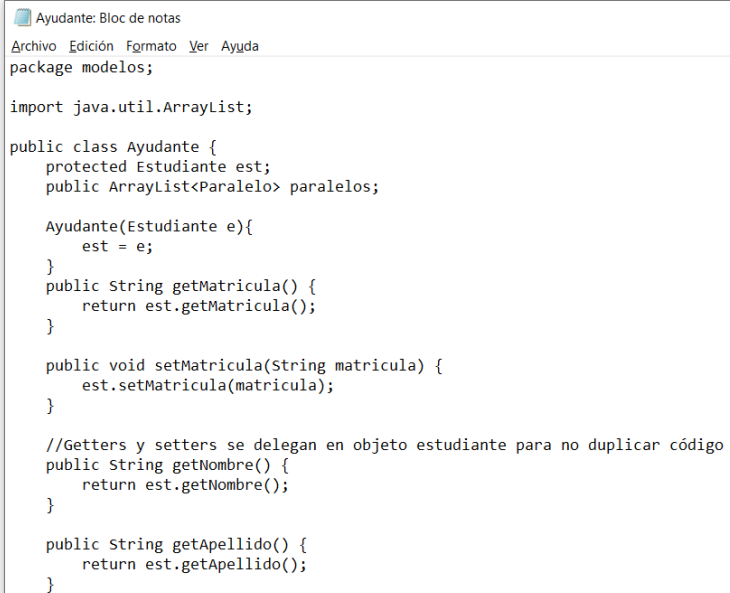
    public String getMatricula() {
        return matricula;
    }

    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;
    }
}
```



```
Ayudante: Bloc de notas
Archivo Edición Formato Ver Ayuda
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();
    }
}
```

Código modificado:

```
package modelos;

import java.util.ArrayList;

public class Ayudante {
    protected Estudiante est;
    public ArrayList<Paralelo> paralelos;

    Ayudante(Estududiante e){
        est = e;
    }
    public Estudiante getEstudiante(){
        return this.est;
    }
    //Los paralelos se añaden/eliminan directamente del Arraylist de paralelos

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

Consecuencias:

De mantener el código así, las clases serían más extensas y tendrían varios métodos que hacen prácticamente lo mismo.

Técnicas de refactorización: Move Method, Add Parameter and Parameterize Method

Data class

Código original:

<pre>package modelos; public class InformacionAdicionalProfesor { public int añosdeTrabajo; public String facultad; public double BonoFijo; }</pre>	<pre>package modelos; public class Materia { public String codigo; public String nombre; public String facultad; public double notaInicial; public double notaFinal; public double notaTotal; }</pre>
--	--

Código modificado:

```
package modelos;
import java.util.ArrayList;
public class Materia {
    private String codigo;
    private String nombre;
    private String facultad;
    private double notaInicial;
    private double notaFinal;
    private double notaTotal;
    private ArrayList<Paralelo> paralelos;
    //Calcula y devuelve la nota teorica
    public double notaTeorico(double nexamen, double ndeberes, double nlecciones) {
        return (nexamen + ndeberes + nlecciones) * 0.80;
    }
    //Devuelve true o false si esta dentro de ese paralelo
    public boolean isParalelo(Paralelo p) {
        for (Paralelo par : paralelos) {
            if (p.equals(par)) {
                return true;
            }
        }
        return false;
    }
    //Calcula y devuelve la nota contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula.
    public double calcularNota(Paralelo p, double notaTeorico, double ntalleres) {
        double nota = 0;
        if (isParalelo(p)) {
            double notaPractico = (ntalleres) * 0.20;
            nota = notaTeorico + notaPractico;
        }
        return nota;
    }
    //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) {
        if (isParalelo(p)) {
            notaTotal = (p.getMateria().notaInicial + p.getMateria().notaFinal) / 2;
        }
        return notaTotal;
    }
}
```

Consecuencias:

Mantener el código así haría más probable la duplicación de código, además que tendría datos particulares por todo el código en vez de tenerlos en un solo lugar, por lo que haría al código más difícil de comprender.

Técnicas de refactorización: Move method

Feature envy

Código original:

```
package modelos;

public class calcularSueldoProfesor {

    public double calcularSueldo(Profesor prof){
        double sueldo=0;
        sueldo= prof.info.añosdeTrabajo*600 + prof.info.BonoFijo;
        return sueldo;
    }
}
```

Código modificado:

```
package modelos;

import java.util.ArrayList;

public class Profesor extends Persona {

    private String codigo;
    private String nombre;
    private String apellido;
    private int edad;
    private String direccion;
    private String telefono;
    private InformacionAdicionalProfesor info;
    private ArrayList<Paralelo> paralelos;

    public Profesor(String codigo, String nombre, String apellido, String facultad, int edad, String direccion, String telefono) {
        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 double calcularSueldo(Profesor prof) {
        double sueldo = 0;
        sueldo = prof.info.getAñosdeTrabajo() * 600 + prof.info.getBonoFijo();
        return sueldo;
    }
}
```

Consecuencias:

Mantener el código así haría que este esté mal organizado, además que aumenta la probabilidad de duplicación de código.

Técnicas de refactorización: Move Method








Lazy class

Código original:

```
package modelos;

public class calcularSueldoProfesor {
}
```

Código modificado:

-  Ayudante
-  Estudiante
-  InformacionAdicionalProfesor
-  Materia
-  Paralelo
-  Persona
-  Profesor

Consecuencias:

Mantener clases consume recursos y tiempo, por lo que mantenerla complicaría el mantenimiento, además extendería el código.

Técnicas de refactorización: Inline Class

Inappropriate Intimacy

Código original:

```
public class InformacionAdicionalProfesor {  
    public int añosdeTrabajo;  
    public String facultad;  
    public double BonoFijo;  
}  
  
public class calcularSueldoProfesor {  
    public double calcularSueldo(Profesor prof) {  
        double sueldo=0;  
        sueldo= prof.info.añosdeTrabajo*600 + prof.info.BonoFijo;  
        return sueldo;  
    }  
}
```

Código modificado:

```

package modelos;

public class InformacionAdicionalProfesor {
    protected int añosdeTrabajo;
    protected String facultad;
    protected double BonoFijo;

    public int getAñosdeTrabajo() {
        return añosdeTrabajo;
    }

    public void setAñosdeTrabajo(int añosdeTrabajo) {
        this.añosdeTrabajo = añosdeTrabajo;
    }

    public String getFacultad() {
        return facultad;
    }

    public void setFacultad(String facultad) {
        this.facultad = facultad;
    }

    public double getBonoFijo() {
        return BonoFijo;
    }

    public void setBonoFijo(double BonoFijo) {
        this.BonoFijo = BonoFijo;
    }
}

package modelos;

public class calcularSueldoProfesor {

    public double calcularSueldo(Profesor prof){
        double sueldo=0;
        sueldo= prof.info.getAñosdeTrabajo()*600 + prof.info.BonoFijo;
        return sueldo;
    }
}

```

Consecuencias:

Mantener el código así complicaría el soporte y la reutilización de código, además se tendría un código más desorganizado y menos seguro.

Técnicas de refactorización: Hide Delegate

Duplicated Code

Código original:

```
//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 notaTeorico, double ntalleres) {
    double notaInicial = 0;
    if (isParalelo(p)) {
        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 notaTeorico, double ntalleres) {
    double notaFinal = 0;
    if (isParalelo(p)) {
        double notaPractico = (ntalleres) * 0.20;
        notaFinal = notaTeorico + notaPractico;
    }
    return notaFinal;
}
```

Código modificado:

```
package modelos;
import java.util.ArrayList;
public class Materia {
    public String codigo;
    public String nombre;
    public String facultad;
    public double notaInicial;
    public double notaFinal;
    public double notaTotal;
    public ArrayList<Paralelo> paralelos;
    //Calcula y devuelve la nota teorica
    public double notaTeorico(double nexamen, double ndeberes, double nlecciones) {
        return (nexamen + ndeberes + nlecciones) * 0.80;
    }
    //Devuelve true o false si esta dentro de ese paralelo
    public boolean isParalelo(Paralelo p) {
        for (Paralelo par : paralelos) {
            if (p.equals(par)) {
                return true;
            }
        }
        return false;
    }
}
//Calcula y devuelve la nota contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula.
public double CalcularNota(Paralelo p, double notaTeorico, double ntalleres) {
    double nota = 0;
    if (isParalelo(p)) {
        double notaPractico = (ntalleres) * 0.20;
        nota = notaTeorico + notaPractico;
    }
    return nota;
}
//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) {
    if (isParalelo(p)) {
        notaTotal = (p.getMateria().notaInicial + p.getMateria().notaFinal) / 2;
    }
    return notaTotal;
}
}
```

Consecuencias:

Mantener el código así haría que exista código duplicado por lo cual la clase crecería sin necesidad, se volvería mas complejo y as costoso de mantener.

Técnicas de refactorización: Extract Method