ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN
DISEÑO DE SOFTWARE
TALLER 8 REFACTORING

PARALELO 102

INTERGRANTES:

DAYANA CUMBA

ROBERT SÁNCHEZ

ALEXANDER PAUTA

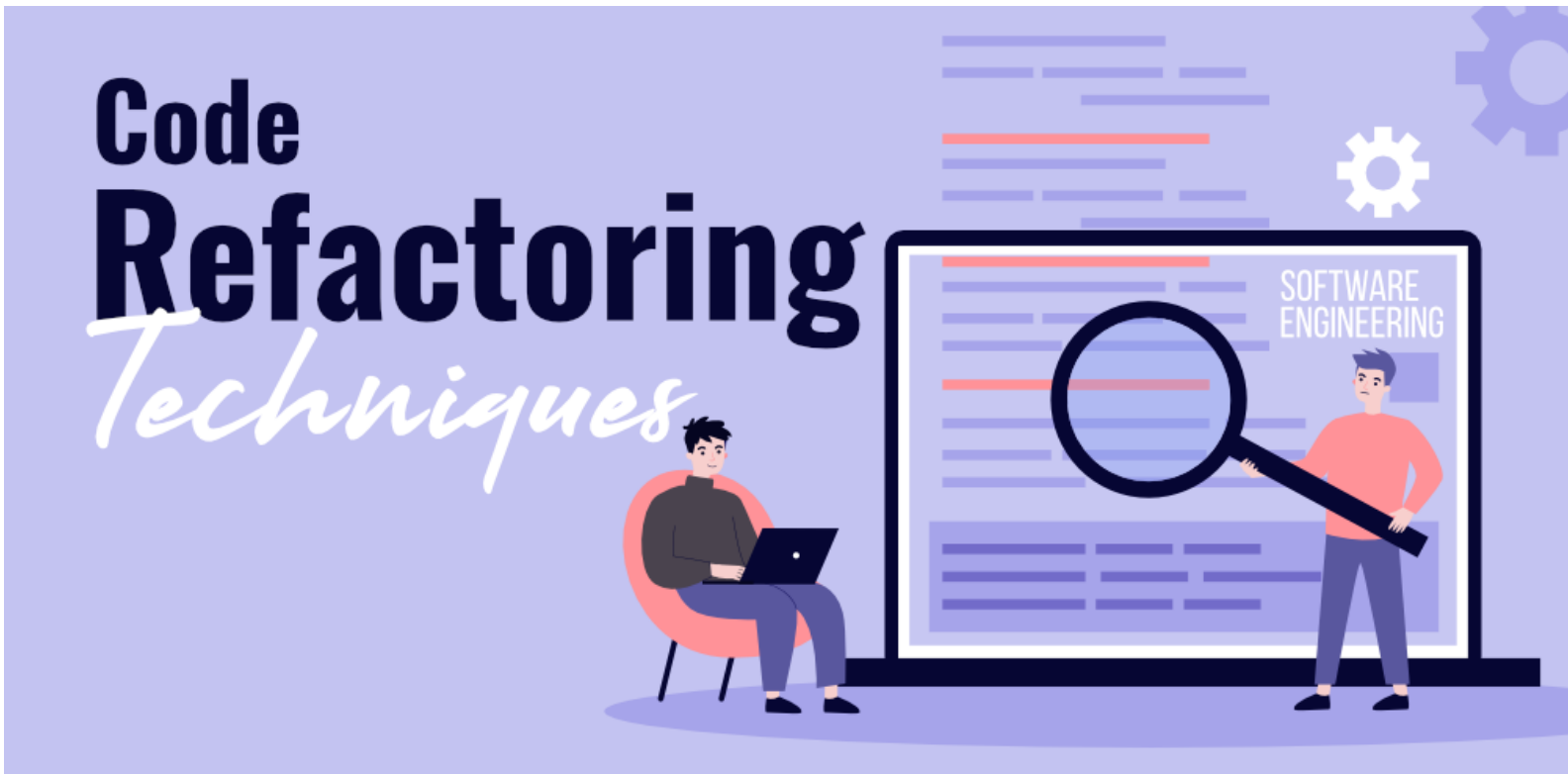

Tabla de contenido

1. Code smell: Inapropiate Intimacy

En este caso la clase ayudante necesita acceder a los parámetros de la clase Estudiante para su funcionamiento generando un acoplamiento entre clases.

Consecuencias:

Esto puede generar que el desarrollador no pueda entender de una manera correcta ninguna de las dos clases. En caso de que se desee realizar un cambio en la clase Estudiante afectaría directamente a la clase Ayudante.

Captura inicial:

```java
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();
    }
```

```java
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;
```

Técnicas de refactorización:

En este caso lo más optimo seria extraer la clase y hacer uso de la herencia, se toma en cuenta al ayudante como un caso “especial” de estudiante haciendo que extienda directamente de dicha clase.

Captura final:

```
public class Ayudante extends Estudiante {

    private ArrayList<Paralelo> ayudantias;

    public Ayudante(ArrayList<Paralelo> ayudantias, String matricula, String facultad, String nombre, String apellido, int
        super(matricula, facultad, nombre, apellido, edad, paralelos, direccion, telefono);
        this.ayudantias = ayudantias;
    }

    //Metodo para imprimir los paralelos que tiene asignados como ayudante
    public void MostrarParalelos() {
        for (Paralelo ayu : ayudantias) {
        }
    }
}
```

2. Code Smell: Long Parameter List

Consecuencias:

Los métodos para calcular notas tienen demasiados parámetros lo que hace que el método no sea entendible.

Captura inicial:

```
//Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se cal
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 calcu

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 c
public double CalcularNotaTotal(Paralelo p){
    double notaTotal=0;
    for(Paralelo par:paralelos){
        if(p.equals(par)){
            notaTotal=(p.getMateria().notaInicial+p.getMateria().notaFinal)/2;
```

Técnicas de refactorización:

Lo ideal es crear una clase que se encargue de maneja cada parámetro del método calcular notas, para que sea más sencillo acceder a ellas y evitar la sobrecarga en el método.

Captura final:

Nota.java

```
public class Nota {
        private double nexamen;
    private double ndeberes;
    private double nlecciones;
    private double ntalleres;
    private ArrayList<Paralelo> paralelos;

    public double calcularNota(Paralelo p) {
        double notaTeorico=0;
        double notaPractico=0;
        for(Paralelo par:paralelos){
                if(p.equals(par)) {
                        notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
                notaPractico=(ntalleres)*0.20;

                }
        }
        return notaTeorico+notaPractico;

    }
}
```

Estudiante.java

```
//Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El te
public double CalcularNotaInicial(Paralelo p, Nota n) {
    double notaInicial = n.calcularNota(p);
    return notaInicial;
}

//Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teor
public double CalcularNotaFinal(Paralelo p, Nota n) {
    double notaFinal = n.calcularNota(p);
    return notaFinal;
}

//Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. Esta
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;

}
```

3. Code Smells: Data Clumps

Consecuencias:

Se tendrían variables repetidas como el nombre y apellidos entre las clases de estudiante y profesor incluido los métodos getters y setters por lo que tendríamos un código extenso.

- **Captura inicial:**

```
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;
```

Técnicas de refactorización:

Para mejorarlo creamos una clase Persona que contiene las variables y métodos que se repetían en ambas clases.

- **Captura final:**

```java
public class Persona {

    protected String nombre;
    protected String apellido;
    protected int edad;
    protected ArrayList<Paralelo> paralelos;
    protected String direccion;
    protected String telefono;

    public Persona(String nombre, String apellido, int edad, ArrayList<Paralelo> paralelos, String direccion, String telefono) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.edad = edad;
        this.paralelos = paralelos;
        this.direccion = direccion;
        this.telefono = telefono;
    }

    public String getNombre() {
        return nombre;
    }

    public String getApellido() {
        return apellido;
    }

    public int getEdad() {
        return edad;
    }

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

    public String getDireccion() {
        return direccion;
```

4. Code Smells: Tempory Field

Consecuencias:

Los campos temporales obtienen sus valores (y, por lo tanto, los objetos los necesitan) solo en determinadas circunstancias. Fuera de estas circunstancias, están vacías. Variables que no tienen sentido crearlas.

Captura inicial:

```
public class calcularSueldoProfesor {

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

Técnicas de refactorización:

Podemos eliminar las variables innecesarias y poner el código en una sola sentencia.

Captura final:

```
public class calcularSueldoProfesor {

    public double calcularSueldo(Profesor prof){
        return  prof.getInfo().añosdeTrabajo*600 + prof.getInfo().BonoFijo;
    }
}
```

5. Code Smells: Comments

Consecuencias:

El código está lleno de comentarios innecesarios en métodos muy obvios cuando solo se deben usar en métodos complejos.

Captura inicial:

```
//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;
}
```

Técnicas de refactorización:

Se eliminan los comentarios innecesarios de los getters y setters solo se conservan en los algunos métodos medianamente complejos.

Captura final:

```
public String getNombre() {
    return nombre;
}

public String getApellido() {
    return apellido;
}

public int getEdad() {
    return edad;
}

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

public String getDireccion() {
    return direccion;
}

public String getTelefono() {
    return telefono;
}
```

6. Code Smells: Lazy Class

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.

Captura inicial:

```
public class calcularSueldoProfesor {

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

Técnicas de refactorización:

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.

Captura final:

```
public class Profesor extends Persona {

    private String codigo;

    private InformacionAdicionalProfesor info;

    public InformacionAdicionalProfesor getInfo() {
        return info;
    }

    public Profesor(String codigo, String nombre, String apellido, int edad, ArrayLis
        super(nombre, apellido, edad, paralelos, direccion, telefono);
        this.codigo = codigo;
    }

    public void anadirParalelos(Paralelo p) {
        paralelos.add(p);
    }

    public double calcularSueldo(Profesor prof) {
        return prof.getInfo().añosdeTrabajo * 600 + prof.getInfo().BonoFijo;
    }

}
```