Memoria Proyecto Global Diseño de Software: Refactorización

Por: Gonzalo Fernández Suárez

1. Implementar al menos una refactorización de tipo **extracción de método.**

“Se tiene un fragmento de código que se puede agrupar moviéndolo a un nuevo método separado y se reemplaza el código anterior con una llamada al método”

En el paquete enemigos la clase *FabricaEnemigos* contiene un único método *creaEnemigo()* que se puede refactorizar dividiendo su código en dos mitades, y cada una insertada en un método distinto de esta misma clase

ANTES:

FabricaEnemigos.java

```
1  package Enemigos;
2
3  public class FabricaEnemigos implements Fabrica {
4
5      public Enemigo creaEnemigo() {
6
7          //Los atributos de cada enemigo se generan de forma aleatoria
8
9          int vida = (int) Math.floor( Math.random()*199 + 1 );  //Numero aleatorio entre 1-200
10         int skillMagico = 200 - vida;
11         int fuerza = (int) Math.floor( Math.random()*199 + 1 );  //Numero aleatorio entre 1-200
12         int velocidad = 200 - fuerza;
13
14         int boss = (int) Math.floor( Math.random()*4 + 1); //Numero aleatorio entre 1-5 (%20)
15
16         if (boss == 1 ) {
17             vida = 300;//Los bosses tienen obligatoriamente 300 de vida
18             fuerza += 50;
19             return new Boss (vida,skillMagico,fuerza,velocidad);
20         }
21         else {
22             //Dependiendo de los atributos sera un tipo u otro
23             if (skillMagico > fuerza) {
24                 skillMagico +=30;
25                 return new Mago (vida,skillMagico,fuerza,velocidad);
26             }
27             if (skillMagico < fuerza ) {
28                 fuerza += 30;
29                 return new Guerrero (vida,skillMagico,fuerza,velocidad);
30             }
31             else { //skillMagico == fuerza
32                 skillMagico += 100;
33                 fuerza += 100;
34                 return new Brujo (vida,skillMagico,fuerza,velocidad);
35             }
36         }
37
38     }
39
40
41 }//Fin clase FabricaEnemigos
42
```

DESPUES:

```
public Enemigo creaEnemigo() {

    //Los atributos de cada enemigo se generan de forma aleatoria
    int vida =0,skillMagico =0,fuerza =0,velocidad =0;
    generaAtributosRandom(vida,skillMagico,fuerza,velocidad);

    //Genera el Enemigo en Funcion de sus Atributos
    return creaPersonaje(vida,skillMagico,fuerza,velocidad);

}
```

```
public void generaAtributosRandom(int vida,int skillMagico,int fuerza, int velocidad) {
    vida = (int) Math.floor( Math.random()*199 + 1 );  //Numero aleatorio entre 1-200
    skillMagico = 200 - vida;
    fuerza = (int) Math.floor( Math.random()*199 + 1 );  //Numero aleatorio entre 1-200
    velocidad = 200 - fuerza;
}

public Enemigo creaPersonaje (int vida,int skillMagico,int fuerza, int velocidad) {

    int boss = (int) Math.floor( Math.random()*4 + 1); //Numero aleatorio entre 1-5 (%20)

    if (boss == 1 ) {
        vida = 300;//Los bosses tienen obligatoriamente 300 de vida
        fuerza += 50;
        return new Boss (vida,skillMagico,fuerza,velocidad);
    }
    else {
        //Dependiendo de los atributos sera un tipo u otro
        if (skillMagico > fuerza) {
            skillMagico +=30;
            return new Mago (vida,skillMagico,fuerza,velocidad);
        }
        if (skillMagico < fuerza ) {
            fuerza += 30;
            return new Guerrero (vida,skillMagico,fuerza,velocidad);
        }
        else { //skillMagico == fuerza
            skillMagico += 100;
            fuerza += 100;
            return new Brujo (vida,skillMagico,fuerza,velocidad);
        }
    }

}
```

He creado dos nuevos métodos, cada uno hace una de las partes de la creación del enemigo, y los llamo en *creaEnemigo()*.

2. Implementar al menos una refactorización de tipo **mover método.**

“Cuando un método se usa más en otra clase que en su propia clase, la solución será crear un nuevo método en la clase que lo utilice más.”

En el paquete *Jugador* en la clase *JugadorPrincipal* tenemos el método *generarAtributos()* que solo se llama en la clase *Singleton*, por lo que lo voy a mover a la esta clase.

ANTES:
Clase *JugadorPrincipal*

```
public void generarAtributos() {

    Scanner sc = new Scanner(System.in);

    System.out.print(" ¡Bienvenido! \n ¡Ha llegado la hora de crear tu personaje!\n ");
    System.out.print("\nPuedes asignar 200 puntos entre VIDA y FUERZA MAGICA ¡Adelante! \nVida : ");
    int vida = sc.nextInt();

    while (vida < 1 || vida > 200 ) {
        System.out.print("Introduce un numero entre [1-200] por favor ");
        vida = sc.nextInt();
    }

    int skillMagica = 200 - vida;
    System.out.print("\nEntonces tu Vida sera: "+vida+"     y tu Fuerza Magica sera: "+skillMagica+"\n");

    System.out.print("\nAhora puedes asignar 200 puntos entre FUERZA y VELOCIDAD ¡Adelante! \nFuerza : ");
    int fuerza = sc.nextInt();

    while (fuerza < 1 || fuerza > 200 ) {
        System.out.print("Introduce un numero entre [1-200] por favor ");
        fuerza = sc.nextInt();
    }

    int velocidad = 200 - fuerza;
    System.out.print("\nEntonces tu Fuerza sera: "+fuerza+"     y tu Velocidad sera: "+velocidad+"\n");
    System.out.print("\n¡PERFECTO!");

    //Tras obtener los datos, se incorporan al jugador
    this.vida = vida;
    this.skillMagica = skillMagica;
    this.fuerza = fuerza;
    this.velocidad = velocidad;

}
```

Clase Singleton

Singleton.java | Brujo.java | PosicionDefe... | Caracteristi...

```
1  package Singleton;
2
3  import java.util.Scanner;
8
9  public class Singleton {
10
11     private int contador = 0;
12     private static Singleton instancia = null;
13
14     private Singleton() {}
15
16     public static Singleton getInstance() {
17         if (instancia == null) {
18             instancia = new Singleton();
19         }
20         return instancia;
21     }
22
23     public void consola() {
24         JugadorPrincipal p1 = new JugadorPrincipal();
25         p1.generarAtributos();
26         //Funciona
27         p1.generarDecoracion(p1);
28         //Funciona
29         while (p1.getVida() > 0) {
30             combate(p1);
31         }
32
33     }//Fin metodo CONSOLA
```

DESPUES:
Clase *Singleton*

```
public class Singleton {

    private int contador = 0;
    private static Singleton instancia = null;

    private Singleton() {}

    public static Singleton getInstance() {
        if (instancia == null) {
            instancia = new Singleton();
        }
        return instancia;
    }

    public void consola() {
        JugadorPrincipal p1 = new JugadorPrincipal();
        generarAtributos(p1);
        //Funciona
        p1.generarDecoracion(p1);
        //Funciona
        while (p1.getVida() > 0) {
            combate(p1);
        }

    }//Fin metodo CONSOLA
```

Metodo *generarAtributos(...)* movido a la clase *Singleton*

```
public void generarAtributos(JugadorPrincipal p) {

        Scanner sc = new Scanner(System.in);

        System.out.print(" ¡Bienvenido! \n ¡Ha llegado la hora de crear tu personaje!\n ");
        System.out.print("\nPuedes asignar 200 puntos entre VIDA y FUERZA MAGICA ¡Adelante! \nVida : ");
        p.setVida ( sc.nextInt());

        while (p.getVida() < 1 || p.getVida() > 200 ) {
            System.out.print("Introduce un numero entre [1-200] por favor ");
            p.setVida ( sc.nextInt());
        }

        p.setSkillMagica (200 - p.getVida());
        System.out.print("\nEntonces tu Vida sera: "+p.getVida()+"      y tu Fuerza Magica sera: "+p.getSkillMagica()+"\n");

        System.out.print("\nAhora puedes asignar 200 puntos entre FUERZA y VELOCIDAD ¡Adelante! \nFuerza : ");
        p.setFuerza( sc.nextInt());

        while (p.getFuerza() < 1 || p.getFuerza() > 200 ) {
            System.out.print("Introduce un numero entre [1-200] por favor ");
            p.setFuerza( sc.nextInt());
        }

        p.setVelocidad (200 - p.getFuerza());
        System.out.print("\nEntonces tu Fuerza sera: "+p.getFuerza()+"      y tu Velocidad sera: "+p.getVelocidad()+"\n");
        System.out.print("\n¡PERFECTO!");

    }
```

3. Implementar al menos una refactorización de tipo **descomponer condicional.**

   “Tenemos una declaración complicada en el condicional. Extraemos métodos de la condición y del cuerpo condicional.”

En la clase *Singleton* tengo el metodo *calculadoraCombate* que se encarga de todos los cálculos de ataques en las peleas. Este método es totalmente condicional, por lo que lo voy a refactorizar descomponiendo el condicional.

ANTES:

```
public void calculadoraCombate(JugadorPrincipal p, Enemigo e, int turno,int bonificacion) {

    String estrategia = escogeEstrategia(turno,e);


    int v1,v2,v3,v4;//Variables usadas para los calculos;


    if (turno%2 != 0) { //Turno del jugador

        switch (estrategia) {
        case "directo":
            v1 = p.getFuerza() + bonificacion;
            v2 = e.getFuerza();
            v3 = v1 - v2;
            v4 = e.getVida();
            if (v3 <= 0) {
                System.out.print("\n¡El enemigo es inmune a ese ataque!\n");
                break;
            }
            else {
                e.setVida(v4-v3);
                System.out.print("\nENEMIGO -> -"+v3+" de vida");
                break;
            }
        case "magico":
            v1 = p.getSkillMagica() + bonificacion;
            v2 = e.getSkillMagica();
            v3 = v1 -v2;
            v4 = e.getVida();
            if (v3 <= 0) {
                System.out.print("\n¡El enemigo es inmune a ese ataque!\n");
                break;
            }
            else {
                e.setVida(v4-v3);
                System.out.print("\nENEMIGO -> -"+v3+" de vida");
```

… Ocupa 131 líneas de código, es muy largo, y todos son if, else, switch case, etc...

DESPUES:

```
public void calculadoraCombate(JugadorPrincipal p, Enemigo e, int turno,int bonificacion) {

    String estrategia = escogeEstrategia(turno,e);


    int v1=0,v2=0,v3=0,v4=0;//Variables usadas para los calculos;


    if (turno%2 != 0) { //Turno del jugador

        turnoDelJugador(estrategia, v1, v2, v3, v4,  p,  bonificacion, turno,  e);


    }
    else { //Turno del enemigo
        turnoDelEnemigo(estrategia, v1, v2, v3, v4,  p,  bonificacion, turno,  e);

    }

}//Fin metodo calculadoraCombate
```

Ahora ocupa 19 líneas de código y es mucho más legible. He encapsulado las distintas partes en métodos aparte. Y dentro de estos, he ido encapsulando las distintas partes condicionales y los fragmentos de código en métodos más pequeños e independientes.

4. Implementar al menos una refactorización de tipo **subir campo.**

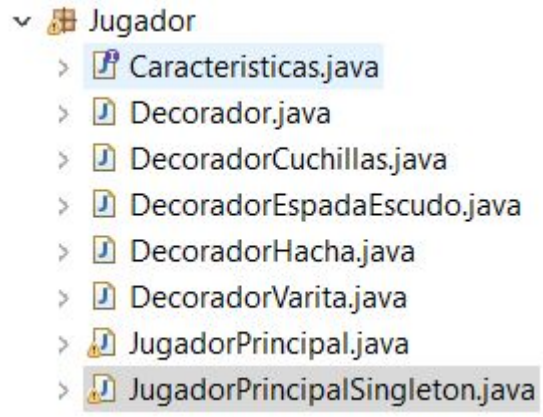
“Si dos clases tienen el mismo campo, hay que quitar el campo de las subclases y moverlo a la superclase.”

Recuerdo que mientras programaba el proyecto, en el paquete enemigo todas las subclases (Mago,guerrero,monstruo…) tenían que tener una String que contuviese su tipo, por ejemplo: ”Mago” , y declaré directamente la String en la clase padre *Enemigo*. No he encontrado nada donde hacerlo ahora. Creo que este método de refactorización es algo que se hace muchas veces por sentido común cuando programas.

5. Implementar al menos una refactorización de tipo **extraer subclase.**

“Si una clase tiene propiedades que solo son usadas en determinadas instancias, se crea una subclase para dicho grupo de propiedades.”

En la clase *JugadorPrincipal* tengo métodos que solo se usan en la clase *Singleton*, es decir, solo los usa la instancia *Singleton*. Por lo que he generado una subclase *JugadorPrincipalSingleton* y he introducido en su interior todo lo que uso con mi instancia *Singleton.*




```java
public class JugadorPrincipalSingleton {

    public void generarDecoracion(JugadorPrincipal p) {
        Scanner sc = new Scanner(System.in);

        System.out.print("\n¡Es el momento de elegir tu arma!\n");
        System.out.print("[1] Espada y Escudo (+40% Vida, +20% Fuerza)\n");
        System.out.print("[2] Hacha de guerra (+50% Fuerza)\n");
        System.out.print("[3] Varita (+60% Fuerza Magica)\n");
        System.out.print("[4] Cuchillas (+60% Velocidad)\n");

        int seleccion = sc.nextInt();

        while (seleccion < 1 || seleccion > 4) {
            System.out.print("\nIntroduce un numero entre [1-4]\n");
            seleccion = sc.nextInt();
        }

        System.out.print("\n¡Es el momento de elegir con que elemento quieres hechizar tu arma!\n ");
        System.out.print("[1] Fuego\n ");
        System.out.print("[2] Agua\n ");
        System.out.print("[3] Tierra\n ");
        System.out.print("[4] Aire\n ");

        int seleccion2 = sc.nextInt();

        while (seleccion2 < 1 || seleccion2 > 4) {
            System.out.print("\nIntroduce un numero entre [1-4]\n");
            seleccion2 = sc.nextInt();
        }


        String elemento = null;

        switch (seleccion2) {
        case 1:
            elemento = "fuego";
            elemento = "fuego";
            break;
        case 2:
```

...