

Grado en Ingeniería de Software



Práctica Refactoring

Sergio Esteban Tarrero Curso - 2019/2020

Diseño de Software





1

Enunciado

2

Desarrollo

1

Enunciado

Se deberá refactorizar el juego de combate realizado en el proyecto de patrones de diseño, a fin de aumentar la calidad y mantenibilidad del diseño. Para ello se aplicarán las refactorizaciones de M. Fowler (1999) vistas en clase. Habrá que entregar una memoria donde se deberá explicar las modificaciones realizadas al juego, y donde aparezca en cada caso el código antes y después de refactorizar, explicando y justificando el uso en cada caso de esas refactorizaciones.

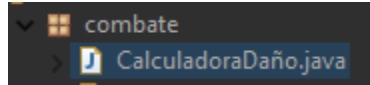
La meta será lograr mejorar el diseño con respecto a la versión anterior. Algunos criterios básicos a valorar son: clases pequeñas, abundantes y con responsabilidades descentralizadas, diseño modular y con alto grado de abstracción, uso adecuado de las relaciones de herencia, empleo de polimorfismo, métodos cortos y ampliamente usables, evitar las clases sin comportamiento, uso adecuado de interfaces, número de campos reducido en cada clase,...

Para cada refactorización aplicada se deberá incluir además del nombre de la técnica de refactorización, indicar el 'mal olor' al que hace frente y dibujar un diagrama de clases y/o fragmentos de código involucrados, con explicación de los mismos. Si por cualquier motivo no se encontrase donde aplicarlas, se podrá incluir cualquier otra técnica del catálogo de M. Fowler.

2

Desarrollo

Calculadora de daño



```
public boolean ardiendo()
{
    int randomNum = ThreadLocalRandom.current().nextInt(0, 100);

    if(randomNum >= 10)
    {
        return true;
    }

    return false;
}
```

Estado **ardiendo**

```
public boolean congelado()
{
    int randomNum = ThreadLocalRandom.current().nextInt(0, 100);

    if(randomNum >= 15)
    {
        return true;
    }

    return false;
}
```

Estado **congelado**

```
public boolean desorientado()
{
    int randomNum = ThreadLocalRandom.current().nextInt(0, 100);

    if(randomNum >= 25)
    {
        return true;
    }

    return false;
}
```

Estado **desorientado**

```
public boolean sangrando()
{
    int randomNum = ThreadLocalRandom.current().nextInt(0, 100);

    if(randomNum >= 20)
    {
        return true;
    }

    return false;
}
```

Estado **sangrando**

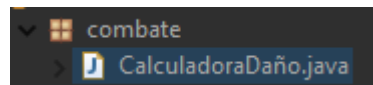
```
public boolean envenenado()
{
    int randomNum = ThreadLocalRandom.current().nextInt(0, 100);

    if(randomNum >= 15)
    {
        return true;
    }

    return false;
}
```

Estado **envenenado**

Calculadora de daño **refactorizada**



Creo una extracción de método:

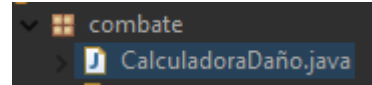
```
private boolean calcularProbabilidadEstado(int probabilidad)
{
    int randomNum = ThreadLocalRandom.current().nextInt(0, 100);

    if(randomNum >= probabilidad)

        return true;

    return false;
}
```


Calculadora de daño **refactorizada**



Los estados pasan a tener esta estructura:

```
public boolean ardiendo()  
{  
    return this.calcularProbabilidadEstado(15);  
}
```

Estado **ardiendo**

```
public boolean congelado()  
{  
    return this.calcularProbabilidadEstado(10);  
}
```

Estado **congelado**

(He reducido la probabilidad del congelado)

```
public boolean desorientado()  
{  
    return this.calcularProbabilidadEstado(25);  
}
```

Estado **desorientado**


```
public boolean sangrando()  
{  
    return this.calcularProbabilidadEstado(20);  
}
```

Estado **sangrando**

```
public boolean envenenado()  
{  
    return this.calcularProbabilidadEstado(15);  
}
```

Estado **envenenado**


Al crearme la función anterior que devuelve un **booleano**, sería sobrecargar el código

>  Homeopata.java

```
if(probabilidadEstado.envenenado() == true)
```




```
if(probabilidadEstado.envenenado())
```

>  Oso.java

```
if(probabilidadEstado.sangrado() == true)
```




```
if(probabilidadEstado.sangrado())
```

>  Terraplanista.java

```
if(probabilidadEstado.desorientado() == true)
```




```
if(probabilidadEstado.desorientado())
```

>  Twittero.java

```
if(probabilidadEstado.ardiendo() == true)
```



```
if(probabilidadEstado.ardiendo())
```

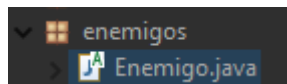
>  Vikingo.java

```
if(probabilidadEstado.congelado() == true)
```



```
if(probabilidadEstado.congelado())
```

Enemigo



Cambio de **protected** a **public** en **habilidadIA()** debido a que lo utilizan las propias clases de enemigo entre ellas

```
protected boolean habilidadIA()
{
    Random rand = new Random();

    int intervaloDeValores = 1;
    int habilidad = rand.nextInt(intervaloDeValores);

    if(habilidad == 1)
        return true;

    return false;
}
```

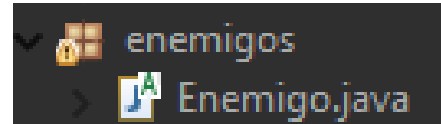


```
public boolean habilidadIA() {
    Random rand = new Random();

    int intervaloDeValores = 1;
    int habilidad = rand.nextInt(intervaloDeValores);

    if(habilidad == 1) {
        return true;
    }
}
```

Se repiten en el mismo método en todas las subclases de enemigo, he extraído estas 4 líneas y he creado una función en la clase padre y todas las subclases llaman ese método



Le tenemos que pasar el **Jugador Objetivo**

```
protected void imprimirFrasesAtaque(Jugador objetivo)
{
    int daño = Personaje.calc.calcularDaño(this, objetivo);

    System.out.println(this.getNombre() + " hace " + String.valueOf(daño) + " daño a " + objetivo.getNombre() + "\n");
    objetivo.dañar(daño);
    System.out.println(objetivo.getNombre() + " le queda " + objetivo.getVida() + "/" + objetivo.getVidaMaxima() + " hp");
}
```

Se repiten en el mismo método en todas las subclases de enemigo, he extraído estas 4 líneas y he creado una función en la clase padre y todas las subclases llaman ese método

> Homeopata.java

```
public void atacar(Jugador objetivo)
{
    System.out.println(this.getNombre() + "Homeópata: jajaja");
    this.imprimirFrasesAtaque(objetivo);
}
```

> Oso.java

```
public void atacar(Jugador objetivo)
{
    System.out.println("Oso: Te meto un zarpazo y no vuelves!");
    this.imprimirFrasesAtaque(objetivo);
}
```

> Terraplanista.java

```
public void atacar(Jugador objetivo)
{
    System.out.println(this.getNombre() + "Tengo pruebas, me he visto todos los vídeos de Oliver Ibáñez!!!!!!");
    this.imprimirFrasesAtaque(objetivo);
}
```

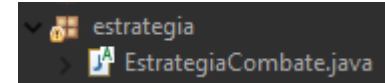
> Twittero.java

```
public void atacar(Jugador objetivo)
{
    System.out.println(this.getNombre() + " mi último tweet ha tenido 16 favs!");
    this.imprimirFrasesAtaque(objetivo);
}
```

> Vikingo.java

```
public void atacar(Jugador objetivo)
{
    System.out.println(this.getNombre() + " Antes de entrar en un lugar, fíjate por dónde se puede salir");
    this.imprimirFrasesAtaque(objetivo);
}
```

Elimino la librería “**import java.util.ArrayList**” debido a que no la voy a necesitar



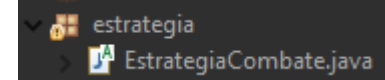
```
1 package estrategia;  
2  
3 import java.util.ArrayList;  
4  
5 import enemigos.Enemigo;  
6 import personajes.Personaje;
```



```
1 package estrategia;  
2  
3 import enemigos.Enemigo;  
4 import personajes.Jugador;  
5 import personajes.Personaje;  
6
```

Método que recibe **Personaje Jugador**

Imprime la frase al atacar de los enemigos

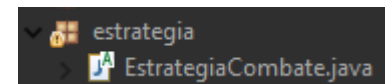


```
public void ejecutar(Personaje jugador)
{
    if(jugador.estaVivo()) {
        System.out.println(this.componente.getNombre() + " se prepara para atacar a " + jugador.getNombre());
        this.componente.atacar((Jugador) jugador);
        System.out.println(this.componente.getNombre() + " va a usar " + this.componente.getnombreHabilidadSecundaria());
        this.componente.usarHabilidadSecundaria((Jugador) jugador);
    }
}
```



```
public abstract void ejecutarEstrategia(Personaje personajesJugador);

protected void atacarJugador(Personaje jugador)
{
    System.out.println(this.componente.getNombre() + " se prepara para atacar a " + jugador.getNombre());
    this.componente.atacar((Jugador) jugador);
}
```



Renombramos otro método:

```
public void ejecutarStrategy(EstrategiaCombate estrategia, Personaje personajesJugador)
{
    estrategia.ejecutar(personajesJugador);
}
```



```
public void ejecutarEstrategia(EstrategiaCombate estrategia, Personaje personajesJugador)
{
    estrategia.ejecutarEstrategia(personajesJugador);
}
```


Entonces hay que renombrar el método en el main:

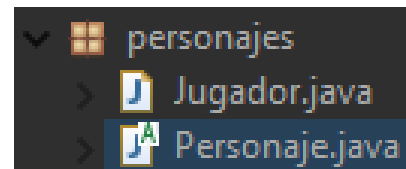


```
enemigos.get(0).ejecutarStrategy(estrategia, jugador);
```



```
enemigos.get(0).ejecutarEstrategia(estrategia, jugador);
```

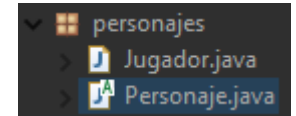
Pasa una clase como parámetro para refactorizar los estados:



```
private boolean getEstado(Class claseEstado)
{
    for(EstadoCombate estado : this.getEstados())
        if(estado.getClass() == claseEstado)
            return true;

    return false;
}
```

Refactorización de los métodos de estados:



```
public boolean getArdiendo() {  
    for(EstadoCombate estado : this.getEstados()) {  
        if(estado.getClass() == EstadoArdiendo.class) {  
            return true;  
        }  
    } return false;  
}
```



```
public boolean getArdiendo() { return this.getEstado(EstadoArdiendo.class); }
```

```
public boolean getCongelado() {  
    for(EstadoCombate estado : this.getEstados()) {  
        if(estado.getClass() == EstadoCongelado.class) {  
            return true;  
        }  
    } return false;  
}
```



```
public boolean getCongelado() { return this.getEstado(EstadoCongelado.class); }
```

```
public boolean getCongelado() {  
    for(EstadoCombate estado : this.getEstados()) {  
        if(estado.getClass() == EstadoCongelado.class) {  
            return true;  
        }  
    } return false;  
}
```



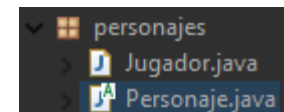
```
public boolean getDesorientado() { return this.getEstado(EstadoDesorientado.class); }
```

```
public boolean getDesorientado() {  
    for(EstadoCombate estado : this.getEstados()) {  
        if(estado.getClass() == EstadoDesorientado.class) {  
            return true;  
        }  
    } return false;  
}
```



```
public boolean getSangrando() { return this.getEstado(EstadoSangrando.class); }
```

Queda con esta estructura:



```
private boolean getEstado(Class claseEstado)
{
    for(EstadoCombate estado : this.getEstados())
        if(estado.getClass() == claseEstado)
            return true;

    return false;
}

public boolean getArdiendo() { return this.getEstado(EstadoArdiendo.class); }
public boolean getCongelado() { return this.getEstado(EstadoCongelado.class); }
public boolean getDesorientado() { return this.getEstado(EstadoDesorientado.class); }
public boolean getSangrando() { return this.getEstado(EstadoSangrando.class); }
```

Métodos que te generan los combates
en cada escenario

Fábrica Desierto

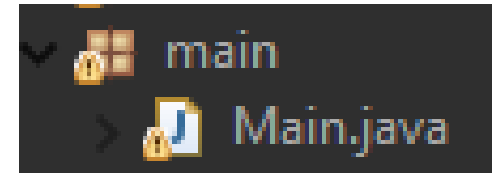
```
public static ArrayList<Enemigo> generarCombateDesierto()  
{  
    ArrayList<Enemigo> enemigos = new ArrayList<Enemigo>();  
  
    FactoryDesierto fabricaDesierto = new FactoryDesierto();  
  
    enemigos.add(fabricaDesierto.getOso());  
    enemigos.add(fabricaDesierto.getTerraplanista());  
    enemigos.add(fabricaDesierto.getTwittero());  
  
    return enemigos;  
}
```

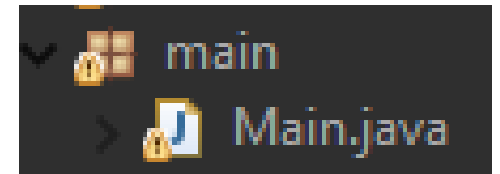
Fábrica Luna

```
public static ArrayList<Enemigo> generarCombateLuna()  
{  
    ArrayList<Enemigo> enemigos = new ArrayList<Enemigo>();  
  
    FactoryLuna fabricaLuna = new FactoryLuna();  
  
    enemigos.add(fabricaLuna.getOso());  
    enemigos.add(fabricaLuna.getTerraplanista());  
    enemigos.add(fabricaLuna.getTwittero());  
    enemigos.add(fabricaLuna.getVikingo());  
  
    return enemigos;  
}
```

Fábrica Mercadona

```
public static ArrayList<Enemigo> generarCombateMercadona()  
{  
    ArrayList<Enemigo> enemigos = new ArrayList<Enemigo>();  
  
    FactoryMercadona fabricaMercadona = new FactoryMercadona();  
  
    enemigos.add(fabricaMercadona.getOso());  
    enemigos.add(fabricaMercadona.getTerraplanista());  
    enemigos.add(fabricaMercadona.getTwittero());  
    enemigos.add(fabricaMercadona.getHomeopata());  
    enemigos.add(fabricaMercadona.getVikingo());  
  
    return enemigos;  
}
```





Fábrica Desierto

```
157     enemigos.addAll(generarCombateDesierto());
```

Fábrica Luna

```
175     enemigos.addAll(generarCombateMercadona());
```

Fábrica Mercadona

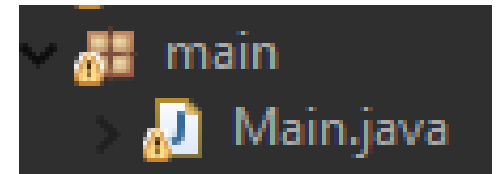
```
175     enemigos.addAll(generarCombateMercadona());
```

Muevo el main al principio para poder llamar a las funciones que necesita el main




```
Main main = new Main();
```


Se cierra el introducir por teclado:




```
190         inputTeclado.close();
```



 Calle Playa de Liencres, 2 bis
(entrada por calle Rozabella)
Parque Europa Empresarial
Edificio Madrid
28290 Las Rozas, Madrid

 900 373 379  info@u-tad.com

 [SOLICITA MÁS INFORMACIÓN](#)



CENTRO ADSCRITO A:

 **Universidad
Camilo José Cela**

PROYECTO COFINANCIADO POR:

