Práctica diseño

Personajes

-Clase personaje:

Para los personajes del juego hemos utilizado una clase principal padre **Personaje** de la cual heredan de ella tanto el **jugador principal** como los **enemigos**. La clase personaje implementa métodos principales comunes como **getters** y **setters**, **tres constructores,** y métodos más específicos como **ejecutarEstados(), añadirEstados()** o **dañar()**.

- **tres constructores** que establecen los valores principales del personaje.
- **ejecutarEstados()** que se usará en el combate y que permitirá observar y ejecutar los estados que se le han añadido al personaje, ya sea sangrado, ardiendo o cualquier tipo de estado.
- **añadirEstados()** que permite añadir un estado a los estados actuales es decir el personaje podrá tener varios estados a la vez.
- **dañar()** que simplemente ataca al personaje objetivo.

```java
public abstract class Personaje
{

    protected String nombre = "SinNombre";

    protected int ataque = 0;
    protected int defensa = 0;

    protected int vida_maxima = 0;
    protected int vida = 0;

    public static CalculadoraDaño calc = new CalculadoraDaño();

    protected ArrayList<EstadoCombate> estados = new ArrayList<EstadoCombate> ();

    public Personaje(String nombre)
    {
        this.nombre = nombre;
    }

    public Personaje(String nombre, int ataque, int defensa)
    {
        this(nombre);
        this.ataque = ataque;
        this.defensa = defensa;
    }

    public Personaje(String nombre, int ataque, int defensa, int vida_maxima)
    {
        this(nombre, ataque, defensa);
        this.vida = vida_maxima;
        this.vida_maxima = vida_maxima;
    }
```

```java
public void añadirEstado(EstadoCombate estado)
{
    boolean flagEstadoEncontrado = false;
    for(EstadoCombate item : this.getEstados())
    {
        if(item.getClass() == estado.getClass())
        {
            item.reiniciarEstado();
            flagEstadoEncontrado = true;
            break;
        }
    }
    if(!flagEstadoEncontrado)
    {
        this.getEstados().add(estado);
    }
}

public void ejecutarEstados()
{
    Iterator<EstadoCombate> itr = this.getEstados().iterator();
    while(itr.hasNext())
    {
        EstadoCombate estado = itr.next();
        if(estado.getEstado()) {
            estado.pasarTurno();
        }
        else {
            itr.remove();
        }
    }
}

public ArrayList<EstadoCombate> getEstados()
{
    return this.estados;
}

public void dañar(int daño)
{
    this.vida = this.vida - daño;
    if(this.vida < 0)
    {
        this.vida = 0;
    }
}
```

-Clase jugador:

La clase **Jugador** a diferencia de la de Personaje **no es abstracta** ya que se debe instanciar, la clase en sí es bastante simple contiene un constructor y el método atacar.

- **constructor** que establece los valores principales de la clase llamando al constructor del padre y un método.
- **atacar()** el cual hace permite atacar al jugador objetivo pero a diferencia de la clase enemigo en este se hace uso del patrón strategy y del decorator esto será explicado más adelante en la parte de combate ya que es donde se definen las estrategias de combate y los decoradores del ataque.

```java
package personajes;

import combate.CombateComponente;

public class Jugador extends Personaje
{

    public Jugador(String nombre, int ataque, int defensa, int vida)
    {
        super(nombre, ataque, defensa, vida);
    }

    public void atacar(Enemigo objetivo)
    {
        CombateComponente ataque = new CombateComponente();
        DecoradorDefensa defensa_decorada = new DecoradorDefensa(ataque);
        DecoradorAtaque ataque_decorado = new DecoradorAtaque(defensa_decorada);

        ataque_decorado.combate(this, objetivo);
    }

}
```

-Clase enemigo:

La clase **enemigo** es una clase **abstracta** ya que a diferencia del jugador no es ella la que se instancia sino que hay varios tipos de enemigos que **heredan** de ella, esta clase ataca de la misma manera que la del jugador exceptuando alguna modificación dependiendo del tipo de personaje y hacemos uso de un **patrón template en ambos** porque cada enemigo podrá hacer uso de dos habilidades una defensiva y otra ofensiva. Se implementa un constructor para establecer los valores principales y los métodos **habilidadIA(), ejecutarAtaque(), atacar(), getters de las habilidades y usarHabilidadPrimaria(), usarHabilidadSecundaria(), ejecutarStrategy().**

- **habilidadIA()** método que establece una probabilidad de que el jugador enemigo pueda ejecutar la primera o segunda habilidad.
- **ejecutarAtaque()** método similar al de atacar del jugador simplemente ataca con el daño base del enemigo.
- **atacar()** método que abstracto que hace uso del patrón template y que deberá ser implementado en las clases hijas.
- **usarHabilidadPrimaria()** método abstracto que define la habilidad primaria del enemigo
- **usarHabilidadSecundaria()** método que define la habilidad secundaria del enemigo

- **ejecutarStrategy()** método que hace uso del patrón strategy y que ejecuta el combate usando una u otra estrategia en el método implementado en la interfaz.

```java
package enemigos;

import java.util.Random;

public abstract class Enemigo extends Personaje
{

    protected String nombreHabilidadPrimaria = "";
    protected String nombreHabilidadSecundaria = "";

    public Enemigo(String nombre, int ataque, int defensa, int vidaMaxima, String nombreHabilidadPrimaria, String nombreHabilidadSecundaria)
    {
        super(nombre, ataque, defensa, vidaMaxima);
        this.nombreHabilidadPrimaria = nombreHabilidadPrimaria;
        this.nombreHabilidadSecundaria = nombreHabilidadSecundaria;
    }

    public void ejecutarAtaque(Jugador objetivo)
    {
        CombateComponente ataque = new CombateComponente();
        DecoradorDefensa defensaDecorada = new DecoradorDefensa(ataque);
        DecoradorAtaque ataqueDecorado = new DecoradorAtaque(defensaDecorada);

        ataqueDecorado.combate(this, objetivo);
    }

    public boolean habilidadIA() {

        Random rand = new Random();

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

        if(habilidad == 1) {

            return true;

        }

        return false;

    }

    public String getNombreHabilidadPrimaria() {
        return this.nombreHabilidadPrimaria;
    }

    public String getnombreHabilidadSecundaria() {
        return this.nombreHabilidadSecundaria;
```

```java
public String getnombreHabilidadSecundaria() {
    return this.nombreHabilidadSecundaria;
}

// Patron Template
public abstract void atacar(Jugador objetivo);

// Lo ultimo si no da tiempo, las habilidades las llaman las estrategias NO el atacar
public abstract void usarHabilidadPrimaria();

public abstract void usarHabilidadSecundaria(Jugador objetivo);

// Patron Strategy
public void ejecutarStrategy(EstrategiaCombate estrategia, Personaje personajesJugador)
{

    estrategia.ejecutar(personajesJugador);

}
```

-Enemigos:

Dentro de las clases de enemigos tenemos a 5 tipos un **oso**, un **homeópata**, un **terraplanista**, un **twittero** y un **vikingo** cada uno con stats diferentes pero sus métodos varían simplemente en la frase que dicen la mecánica de ataque y defensa es la misma. Los métodos que implementa las clases son un **constructor**, **usarHabilidadPrimaria()**, **usarHabilidadSecundaria()**, **atacar()**.

- **constructor** método que permite establecer los valores por defectos.
- **usarHabilidadPrimaria()** Habilidad primaria que cura al enemigo, las habilidades son llamadas desde las estrategias de combate.
- **usarHabilidadSecundaria()** Habilidad secundaria que realiza un ataque que puede provocar un estado en el personaje objetivo o no dependiendo de la probabilidad que salga.
- **atacar()** método que realiza el ataque hacia el personaje objetivo realiza un cálculo del daño y después ataca.

```java
package enemigos;

import personajes.Jugador;

public class Homeopata extends Enemigo
{

    public Homeopata(String nombre, int ataque, int defensa, int vidaMaxima)
    {
        super(nombre, ataque, defensa, vidaMaxima, "Jeringuillazo", "Ataque venenoso illuminati");
    }


    public void usarHabilidadPrimaria()
    {

        System.out.println(this.getNombre() + " se mete un jeringuillazo y se cura 8 hp!");
        this.curar(8);

    }

    public void usarHabilidadSecundaria(Jugador objetivo)
    {

        CalculadoraDaño probabilidadEstado = new CalculadoraDaño();

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

            System.out.println(this.getNombre() + " envenena a " + objetivo.getNombre());
            objetivo.añadirEstado(new EstadoEnvenenado(objetivo));

        }
        else {

            System.out.println(this.getNombre() + " no pudo lanzar su veneno!");

        }


    }


    public void atacar(Jugador objetivo)
    {

        System.out.println(this.getNombre() + " las vacunas causan autismo pero no tanto como el que tengo yo!");
        int daño = Personaje.calc.calcularDaño(this, objetivo);

        System.out.println(this.getNombre() + " hace " + String.valueOf(daño) + " daño a " + objetivo.getNombre());
```

```java
public void atacar(Jugador objetivo)
{

    System.out.println(this.getNombre() + " las vacunas causan autismo pero no tanto como el que tengo yo!");
    int daño = Personaje.calc.calcularDaño(this, objetivo);

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


}
```

Combate

-CalculadoraDaño:

Esta clase se nos ocurrió sobretodo para a la hora de calcular el daño que un enemigo realizaba a otro en vez de tener que meterlo en cada una de las clases lo estructuramos para que esta clase directamente calcule el daño de cualquier personaje y que además calcule la probabilidad de que un estado sea añadido o no.
Implementa los siguientes métodos **calcularDaño(), ardiendo(), congelado(), desorientado(), sangrado() y envenenado()**

- **calcularDaño()** este método realiza la resta del ataque del atacante con la defensa del defensor el resultado será el daño que recibirá el defensor.
- **ardiendo(), congelado(), desorientado(), sangrado() y envenenado()** estos métodos son similares son un simple algoritmo que calcula la posibilidad de que se añada el estado o no.

```java
public class CalculadoraDaño
{

    // daño = (dañoAtacante - defensaDefesor)
    // Patron Singleton
    public int calcularDaño(Personaje atacante, Personaje defensor)
    {
        int daño = atacante.getAtaque() - defensor.getDefensa();

        if(daño < 0)
        {
            daño = 0;
        }

        return daño;

    }

    public boolean ardiendo()
    {

        int randomNum = ThreadLocalRandom.current().nextInt(0, 100);

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

        return false;
    }

    public boolean congelado()
    {

        int randomNum = ThreadLocalRandom.current().nextInt(0, 100);

        if(randomNum >= 30)
        {
            return true;
        }

        return false;
    }
```

-Interfaz de combate:

Interfaz de combate que permite el combate entre un **atacante** y un **defensor**, el sistema de combate hace uso del patrón **decorator**.

```java
package combate;

import personajes.Personaje;

public interface InterfazCombate
{

    public void combate(Personaje atacante, Personaje defensor);

}
```

-DecoradorCombate:

Clase abstracta de la que heredan los decoradores.
Implementa un **constructor** y el método **combate()**.

- **combate()** llama al método combate que tenga implementado el componente.

```java
package combate;

import personajes.Personaje;

public abstract class DecoradorCombate implements InterfazCombate
{

    protected InterfazCombate componente;

    public DecoradorCombate(InterfazCombate componente)
    {
        this.componente = componente;
    }

    public void combate(Personaje atacante, Personaje defensor)
    {
        this.componente.combate(atacante, defensor);
    }
```

-DecoradorAtaque y DecoradorDefensa:

Son clases similares al ser un patrón decorator en la de ataque se dice una frase y en la de defensa otra.
Implementan un constructor y el método combate de la interfaz

```java
package combate;

import personajes.Personaje;

public class DecoradorDefensa extends DecoradorCombate
{

    public DecoradorDefensa(InterfazCombate componente)
    {
        super(componente);
    }

    public void combate(Personaje atacante, Personaje defensor)
    {
        if(defensor.getDefensa() >= 5)
        {
            System.out.println(atacante.getNombre() + " realiza una defensa poderosa!");
        }

        this.componente.combate(atacante, defensor);
    }
}
```

```java
package combate;

import personajes.Personaje;

public class DecoradorAtaque extends DecoradorCombate
{

    public DecoradorAtaque(InterfazCombate componente)
    {
        super(componente);
    }

    public void combate(Personaje atacante, Personaje defensor)
    {
        if(atacante.getAtaque() >= 15)
        {
            /* PLACEHOLDER */
            System.out.println(atacante.getNombre() + " realiza un ataque poderoso!");

        }

        this.componente.combate(atacante, defensor);
    }

}
```

-ComponenteCombate:

Clase que **implementa** a la **interfaz** y es la que realiza el **combate** entre los dos personajes solo implementa un método el cual es llamado en ambos decoradores que es el de **combate()**.

- **combate()** método que realiza el combate entre ambos personajes.

```java
package combate;

import personajes.Personaje;

public class CombateComponente implements InterfazCombate
{

    public void combate(Personaje atacante, Personaje defensor)
    {
        // Este metodo implementa el combate entre los 2 personajes
        int daño = Personaje.calc.calcularDaño(atacante, defensor);

        System.out.println(atacante.getNombre() + " hace " + String.valueOf(daño) + " daño a " + defensor.getNombre());
        defensor.dañar(daño);
        System.out.println(defensor.getNombre() + " le queda " + defensor.getVida() + "/" + defensor.getVidaMaxima() + " hp");
    }

}
```

Estrategias

-EstrategiaCombate:

Clase **abstracta** que hace uso del **patrón strategy** para idear **dos estrategias** que heredan de esta clase tanto **ofensiva** como **defensiva**.
Implementa un constructor y el método **ejecutar().**

- **ejecutar()** método abstracto que implementan las estrategias.

```java
package estrategia;

import enemigos.Enemigo;
import personajes.Personaje;

public abstract class EstrategiaCombate
{

    protected Enemigo componente;

    public EstrategiaCombate(Enemigo componente)
    {
        this.componente = componente;
    }

    // Función que ejecuta la accion de la ia en su turno
    public abstract void ejecutar(Personaje personajesJugador);

}
```

-EstrategiaDefensiva y EstrategiaOfensiva:

Las **estrategias** son las que llaman a las **habilidades** de los personajes y al **atacar** de los personajes si es **defensiva atacará** y se **curará** con la **habilidad primaria** y en el caso de la **ofensiva atacará** pero usará la **habilidad secundaria** para meter algún tipo de **estado**.

```java
package estrategia;

import enemigos.Enemigo;

public class EstrategiaDefensiva extends EstrategiaCombate
{

    public EstrategiaDefensiva(Enemigo componente)
    {
        super(componente);
    }


    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.getNombreHabilidadPrimaria());
            this.componente.usarHabilidadPrimaria();
        }


    }

}
```

```java
package estrategia;

import enemigos.Enemigo;

public class EstrategiaOfensiva extends EstrategiaCombate
{

    public EstrategiaOfensiva(Enemigo componente)
    {
        super(componente);
    }

    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);
        }

    }

}
```

Estados

-EstadoCombate:

Se trata de una clase abstracta que le ocurre lo mismo que a los enemigos en este caso tendremos varios estados.

La clase implementa un constructor y los métodos **pasarTurno(), getters** y **ejecutarEstado().**

- **pasarTurno()** método que ejecuta los estados del personaje y le quita un turno al estado.
- **ejecutarEstado()** método abstracto que se implementa en los demás estados.

```java
package estado;

import personajes.Personaje;

public abstract class EstadoCombate
{

    protected int duracion = -1;
    protected int turnosRestantes = -1;
    protected Personaje componente = null;

    public EstadoCombate(Personaje componente, int duracion)
    {
        this.componente = componente;
        this.duracion = duracion;
        this.turnosRestantes = this.duracion;
    }

    public void pasarTurno()
    {
        this.ejecutarEstado();
        this.turnosRestantes -= 1;
    }

    public boolean getEstado()
    {
        if(this.turnosRestantes > 0)
        {
            return true;
        }
        return false;
    }

    public void reiniciarEstado()
    {
        this.turnosRestantes = this.duracion;
    }

    public int getTurnosRestantes()
    {
        return this.turnosRestantes;
    }

    public abstract void ejecutarEstado();

}
```

-Estados:

Los estados son similares unos dañan y otros simplemente hacen que el jugador no ataque.

```java
package estado;

import personajes.Personaje;

public class EstadoArdiendo extends EstadoCombate
{

    public EstadoArdiendo(Personaje componente)
    {


        super(componente, 3);

    }


    public void ejecutarEstado()
    {

        System.out.println(this.componente.getNombre() + " está ardiendo! Pierde 10 hp");
        System.out.println(this.turnosRestantes + " turnos restantes");
        this.componente.dañar(10);


    }

}
```

Escenarios

-AbstractFactoryEnemigos:

Esta clase es la que permitirá la **generación** de los **enemigos** pero al ser **abstracta** se hacen cargo las **clases** que **heredan** que son los **distintos escenarios**.

```java
package escenarios;

import enemigos.*;

public interface AbstractFactoryEnemigos
{
    public abstract Homeopata getHomeopata();
    public abstract Oso getOso();
    public abstract Terraplanista getTerraplanista();
    public abstract Twittero getTwittero();
    public abstract Vikingo getVikingo();
}
```

-FactoryDesierto, FactoryLuna y FactoryMercadona:

Estas clases son las encargadas de **llamar** a los **constructores** de cada **enemigo** para **generarlos** su estructura es similar.

```java
package escenarios;

import enemigos.Homeopata;

public class FactoryDesierto implements AbstractFactoryEnemigos {

    // Homeopata en el Desierto
    public Homeopata getHomeopata()
    {
        /* Valores Ataque, Defensa y VidaMáxima están de placeholder
        return new Homeopata("Homeopata en el desierto", 5, 5, 100);
    }

    // Oso en el Desierto
    public Oso getOso()
    {
        /* Valores Ataque, Defensa y VidaMáxima están de placeholder
        return new Oso("Oso Caluroso", 8, 12, 100);
    }

    // Terraplanista en el Desierto
    public Terraplanista getTerraplanista()
    {
        /* Valores Ataque, Defensa y VidaMáxima están de placeholder
        return new Terraplanista("Terraplanista en el desierto", 6, 1
    }

    // Twittero en el Desierto
    public Twittero getTwittero()
    {
        /* Valores Ataque, Defensa y VidaMáxima están de placeholder
        return new Twittero("Twittero en el desierto", 6, 8, 100);

    }

    // Vikingo en el Desierto
    public Vikingo getVikingo()
    {
        /* Valores Ataque, Defensa y VidaMáxima están de placeholder
        return new Vikingo("Vikingo en el desierto", 10, 8, 100);
    }

}
```