JAVA – QUIZ SORPRESA!!!!

RUBEN DARIO ORTIZ ORTEGA – P1

**Documentación del Proyecto: Juego con Sistema de Armas y Encantamientos**

**1. Introducción**

Este proyecto es un ejemplo de un sistema de juego donde los personajes pueden blandir armas que están encantadas con diferentes efectos. Se han implementado varios conceptos de diseño orientado a objetos en Java, incluyendo patrones de diseño como Singleton y Factory Method, junto con el uso de abstracción y herencia para organizar la estructura del código.

**2. Requisitos**

El juego cuenta con los siguientes requisitos:

1. **Personajes**:
   * Existen dos tipos de personajes: Héroe y Villano.
   * Solo puede existir un único Héroe durante la ejecución del juego.
   * Pueden existir múltiples instancias de Villanos, pero se limitan a un máximo de 5 instancias por tipo.
2. **Armas y Encantamientos**:
   * Las armas pueden tener diferentes encantamientos.
   * Cada arma, al ser blandidas, activan su encantamiento; al atacar, el encantamiento se aplica; y al soltar el arma, el encantamiento se desactiva.
   * Se implementaron dos tipos de armas (Espada y Martillo) y dos tipos de encantamientos (Fuego y Vorpal).
3. **Armería**:
   * Los personajes pueden comprar armas en una armería.
   * Las armas pueden ser fabricadas por herreros élficos o enanos, dependiendo de la preferencia del cliente.
4. **Gestión de Datos**:
   * Se debe permitir almacenar y mostrar los datos de los personajes y sus armas.

**3. Estructura del Código**

**3.1 Clases Principales**

1. **Personaje (Clase Abstracta)**
   * Clase base para Héroe y Villano.
   * Contiene los atributos comunes nombre y arma.
   * Define métodos abstractos para las acciones comunes: blandir(), atacar(), y soltar().

Java

public abstract class Personaje {

private String nombre;

private Arma arma;

// Métodos y constructores

}

1. **Heroe**
   * Extiende Personaje.
   * Implementa los métodos abstractos definidos en Personaje.
   * Se asegura de que solo exista una instancia (Patrón Singleton).

Java

public class Heroe extends Personaje {

// Implementación de métodos abstractos

}

1. **Villano**
   * Extiende Personaje.
   * También implementa los métodos abstractos definidos en Personaje.
   * Se asegura que haya un límite de 5 instancias por tipo.

Java

public class Villano extends Personaje {

// Implementación de métodos abstractos

}

1. **Arma (Clase Abstracta)**
   * Define las características comunes para las armas.
   * Incluye un atributo encantamiento y métodos abstractos para manejar la activación, aplicación, y desactivación del encantamiento.

Java

public abstract class Arma {

private Encantamiento encantamiento;

// Métodos y constructores

}

1. **Espada y Martillo**
   * Extienden Arma y proporcionan implementaciones para los métodos abstractos.

Java

public class Espada extends Arma {

// Implementación de métodos

}

java

public class Martillo extends Arma {

// Implementación de métodos

}

1. **Encantamiento (Interfaz)**
   * Define los métodos activar(), aplicar(), y desactivar().

Java

public interface Encantamiento {

void activar();

void aplicar();

void desactivar();

}

1. **EncantamientoFuego y EncantamientoVorpal**
   * Implementan la interfaz Encantamiento y proporcionan efectos específicos.

Java

Copiar código

public class EncantamientoFuego implements Encantamiento {

// Implementación de métodos

}

Java

Copiar código

public class EncantamientoVorpal implements Encantamiento {

// Implementación de métodos

}

**3.2 Creación de Armas (Patrón Factory Method)**

La clase Armeria usa el patrón Factory Method para crear armas según la elección del herrero (Elfo o Enano):

Java

public class Armeria {

public Arma crearArma(String tipoHerrero, Encantamiento encantamiento) {

if (tipoHerrero.equalsIgnoreCase("Elfo")) {

return new Espada(encantamiento);

} else if (tipoHerrero.equalsIgnoreCase("Enano")) {

return new Martillo(encantamiento);

} else {

throw new IllegalArgumentException("Tipo de herrero desconocido.");

}

}

}

**3.3 Gestión de Datos y Menú Interactivo**

Se implementó un menú interactivo que permite al usuario realizar varias acciones, como crear personajes, asignar armas, almacenar datos, y mostrar datos existentes.

El almacenamiento se hace en una lista que guarda las instancias de los personajes creados.

Java

public class Main {

public static void main(String[] args) {

// Ejecución del menú interactivo

}

}

**4. Pruebas Unitarias**

Para garantizar que el sistema funcione como se espera, se recomienda el uso de pruebas unitarias con JUnit. A continuación, un ejemplo básico:

Java

import org.junit.Test;

import static org.junit.Assert.\*;

public class JuegoTest {

@Test

public void testCreacionHeroe() {

Heroe heroe = Heroe.getInstance("Conan");

assertNotNull(heroe);

assertEquals("Conan", heroe.getNombre());

}

@Test

public void testAsignacionArma() {

Heroe heroe = Heroe.getInstance("Conan");

Arma espada = new Espada(new EncantamientoFuego());

heroe.asignarArma(espada);

assertNotNull(heroe.getArma());

}

}

**5. Conclusión**

Este proyecto demuestra cómo estructurar un sistema de juego en Java utilizando conceptos de diseño orientado a objetos. Implementa un sistema de creación de personajes y armas con diferentes encantamientos, asegurando un diseño robusto y fácil de mantener. También se proveen mecanismos para probar el modelo mediante pruebas unitarias, garantizando la calidad del software.