Proyecto de temática libre en java.

Luis Ferrer

Contexto del juego:

Este es un proyecto en Java que consta de un Juego de combate por turnos, 1 contra 1, y en el que el jugador controla a un personaje contra la máquina.

La interfaz será únicamente por consola. El jugador seleccionará un personaje entre 4 posibles, y luego eligirá al enemigo, controlado por la computadora, entre los mismos 4 personajes, los cuales son:

- Mago de fuego

- Mago de hielo

- Guerrero con espada

- Guerrero con escudo

Cada personaje en su turno dispondrá de varias acciones:

- Ataque básico

- Habilidad especial

- Curación

La habilidad especial de ambos magos y ambos guerreros es la misma, las dos modifican el "estado" del personaje.

La de los magos envenena al adversario un número de turnos

La de los guerreros aumenta su defensa un número de turnos

El bucle de juego continúa hasta que la salud de uno de los dos llega a 0.

Implementaremos los patrones:

- ***Singleton***: Maneja la instancia única de la configuración del juego o el gestor de estado del juego

- ***Factory*** Method: Crea instancias de diferentes tipos de personajes o entidades

- ***Facade***: Simplifica las interacciones entre la lógica de los componentes del juego.

- ***Strategy***: Permite que los personajes alteren su comportamiento mediante diferentes estrategias implementadas como clases intercambiables.

- ***State***: Gestiona los diferentes estados en los que se encuentra nuestro personaje.

Componentes Abstractos:

- Character: Clase base abstracta para los personajes, define métodos comunes asi como atributos (salud, defensa)

- CharacterFactory: Interfaz para las fabricas de Character.

- CharacterState: Interfaz para los estados que puede adoptar un presonaje (normal, fortificado, envenenado)

- CombatStrategy: Interfaz para las estrategias de combate.

Componentes Concretos:

- Mage, Warrior: Subclases de Character

- FireMage, IceMage: Subclases de Mage

- SwordWarrior, ShieldWarrior: Subclases de Warrior

- AttackStrategy, PoisonStrategy, ArmorStrategy: Estrategias que implementan CombatStrategy

- NormalState, PoisonedState, FortifiedState: Estados concretos para cambiar el comportamiento del personaje durante el juego

- GameManager: Implementa el Singleton para manejar el estado y flujo del juego

- GameFacade: Fachada que proporciona una interfaz simplificada para interactuar con el juego

Implementación del patrón Singleton:

Manejara la configuracion del juego y el estado global, asegurando que solo exista una instancia de cada uno durante la ejecución del juego.

Las clases que lo componen son:

- GameManager: Maneja el estado del juego, los turnos y las reglas globales. Es un singleton para garantizar un único punto de control.

El singleton es un patron de diseño creacional que restringe la instanciación de una clase a un solo objeto.

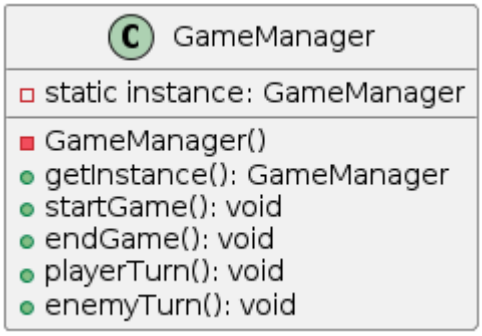
Solo existe una instancia de la clase durante el tiempo de vida de la aplicación. En este caso se crea una clase estática al cargar la clase.

Acceso global a esa instancia mediante getInstance().

El constructor de GameManager es privado para evitar que otras clases instancien,

el método getInstance() en cambio es público.

GameManager como Singleton hace que el estado del juego se maneje de forma centralizada y coherente.



Implementación del patrón Factory Method:

Facilita la creacion de diferentes tipos de personajes, permitiendo añadir nuevos personajes sin modificar el código existente.

Las clases que lo componen son:

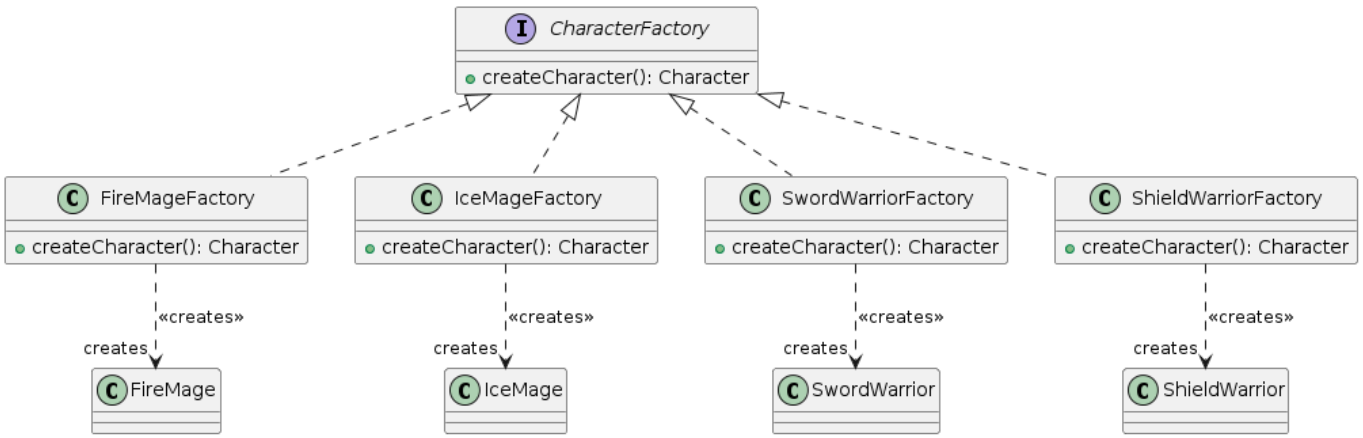
- Interfaz CharacterFactory

- Clases FireMageFactory, IceMageFactory, SwordWarriorFactory, ShieldWarriorFactory implementan la interfaz.

El patrón Factory Method proporciona una interfaz para crear objetos y permite que las subclases alteren el tipo de objetos que se crearán. Mejora la organización del código y facilita su mantenimiento. Reduce las dependencias fuertes entre componentes.

Esta implementación permite extender fácilmente el juego para incluir más tipos de personajes sin modificar el código existente que utiliza las fábricas. Simplemente se pueden agregar más fábricas concretas para los nuevos tipos.

Utilizando el patrón Factory Method, el código que necesita crear instancias de personajes no necesita conocer los detalles de cómo se crean esos personajes ni qué clases concretas están involucradas. Esto facilita la mantenibilidad y la escalabilidad del código.



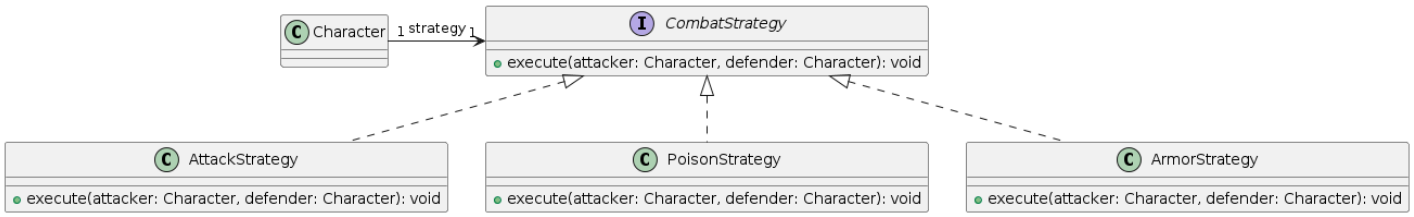
Implementación del patrón Strategy:

Se aplica para definir varios comportamientos o estrategias de combate para los personajes, y permite cambiarlas dinámicamente durante el juego.

Las clases que lo componen son:

- Interfaz CombatStrategy
- Clases ArmorStrategy, AttackStrategy, HealingStrategy, PoisonStrategy

Este patrón hace que se puedan cambiar como atacan o defienden los personajes en tiempo de ejecución. Durante el juego, el método execute() no cambia, lo que cambia es la estrategia que aplica en cada momento el personaje.



Implementación del patrón Facade:

Simplifica las interacciones entre los componentes del juego, entre la interfaz y la lógica del juego.

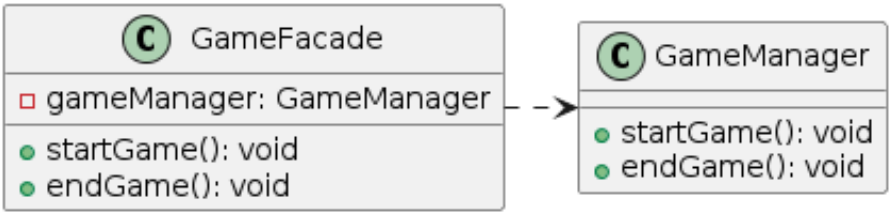
La clase que lo compone es:

- Clase GameFacade

Simplifica las interacciones entre los sistemas del juego, habiendose ya implementado varios patrones de diseño. El patron Facade ayuda a ocultar la complejidad de estos sistemas tras una interfaz mas simple, haciendo mas facil su uso y mantenimiento.

Reduce la complejidad percibida para usuarios y desarrolladores, ayuda a desacoplar el sistema del cliente y los subsistemas complejos, promoviendo una estructura mas robusta.

Proporciona un punto de entrada unico y simple.



Implementación del patrón state:

Permite a un objeto de Character alterar su comportamiento cuando cambia su estado interno. Este patrón maneja los cambios de estado en los personajes, como estar envenenado o fortificado.

Las clases que lo componen son:

- Interfaz CharacterState

- Clases NormalState, PoisonedState y FortifiedState.

El patron gestiona el estado dinámico de los persobajes, como efectos que cambian su comportamiento durante varios turnos. Permite extender el juego con nuevos estados sin alterar la clase principal Character.