Universidad de Los Andes

Miguel Ariza Jimenez – 202010620

Diseño y Programación O.O

Taller 5

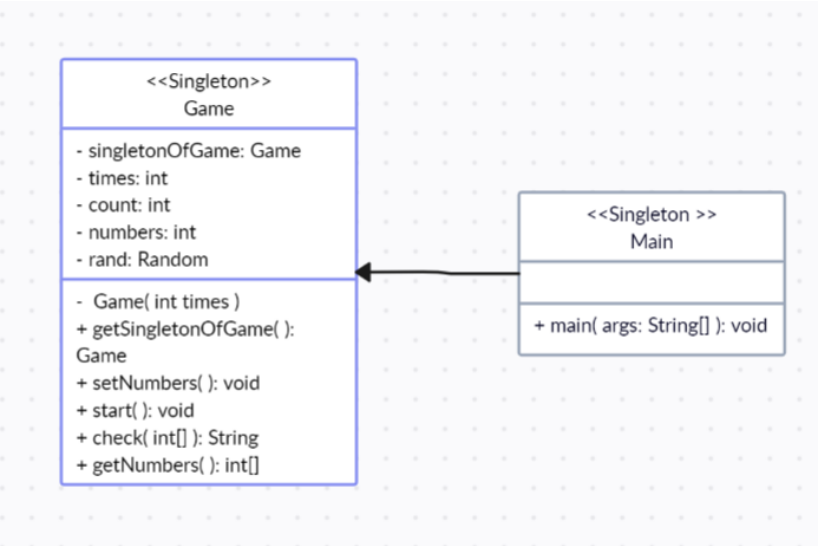
Link del proyecto escogido: https://github.com/kiat/OOP-Design-Patterns.git

Descripción general:

El proyecto seleccionado tiene ejemplos de patrones de GoF hechos en java. Entre ellos están los patrones de comportamiento, estructura y creación. El proyecto tiene uno o varios ejemplos de cada patrón y que finalidad tiene utilizarlo en cada uno. La organización del proyecto esta basada en las clases necesarias para cada patrón y su aplicación. Algunos patrones tienen pruebas ya establecidas, pero no todos. Tambien cabe resaltar que, la complejidad es directamente proporcional a que tan extenso sea un patrón en implementar. Para realizar el análisis de un patrón, se escogió un ejemplo del patrón Singleton, que se ejemplifica con un juego que hace uso de este.

Estructura fragmento seleccionado:

El fragmento seleccionado se basa de una clase con la logica del juego y una clase principal con main para probar el juego. El siguiente diagrama muestra los detalles de los atributos y metodos de las clases.



Para contextualizar, el juego consiste en adivinar un numero de 4 digitos y el jugador siempre inicia con un numero determinado de intentos. Como se puede ver la clase de la logica nombrada Game, se tienen varios metodos que funcionan a partir del patron Singleton. El constructor de la clase crea una unica instancia estatica y privada que se guarda en la variable singletonOfGame, siguiendo el patron Singleton que se explicara más adelante. Asi mismo, existe el metodo getSingletonOfGame que proporciona un punto de acceso global para obtener la instancia unica. El juego se inicializa en el ejemplo con un numero maximo de 10 intentos y se genera un conjunto de 4 numeros aleatorios únicos que se almacenan en el array numbers utilizando el método setNumbers.

El juego empieza con el metodo start, que invoca el metodo getNumbers para obtener los numeros que el usuario pone en consola por medio de un scanner. Para verificar que los numeros ingresados sean los que estan en el array se usa el metodo check. El metodo check funciona comparando los numeros del array con los ingresados y devuelve un resultado en formato xAyB, donde x es la cantidad de numeros correctos en la posición correcta y Y la cantidad de numeros correctos en la posición incorrecta. El juego termina cuando se adivina el numero generado (4A0B) o cuando los intentos se agoten, lo que ocurra primero.

Patrón Singleton:

El patron Singleton es un patron de diseño creacional que garantiza que una clase tenga una sola instancia y proporciona un punto de acceso global a esa instancia. Es uno de los patrones mas simples, pero asi mismo es uno de los mas utiles. Su objetivo principal es controlar cualquier acceso indeseado a la instancia y crear una única instancia de acceso global en todo el sistema. Sus usos habituales son en configuraciones globales de un sistema como una a configuraciones de aplicaciones, registros y manejadores de conexión a bases de datos.

¿Por qué es util usar el patron en el proyecto?

Utilizar el patron en el juego tiene varias ventajas y reduce la dificultad de conectar los metodos de la clase. En primer lugar, al usar el patron es posible crear solo una instancia global del juego, que a su vez solo permite que un juego este en ejecución. Esta instancia tiene un punto de acceso global y su configuración es unica. En segundo lugar, el juego mantiene el mismo estado en toda la aplicación, esto quiere decir que se pueda acceder desde distintas partes del

codigo. Por ultimo, al usar este patron se facilita la extensión de funcionalidades y atributos para la clase principal.

Desventajas de usar el patrón:

En general, existen algunas desventajas que son características del patrón. En primer lugar, el uso del patrón Singleton genera un acoplamiento fuerte en la estructura del código, ya que las partes del sistema dependen directamente de la instancia única. Esto puede hacer que sea más difícil probar y mantener el código. En segundo lugar, la creación de subclases puede ser mas complicado debido al manejo de una única instancia global, pero no significa que sea una complejidad muy alta. Por último, se puede decir que las pruebas unitarias pueden volverse más complicadas debido a la dependencia directa de la instancia única. Puede ser difícil realizar pruebas aisladas de componentes que utilizan el Singleton.

¿De qué otras formas se pudieron haber abordado el problema?

Existen varias maneras de abordar el problema y cada una tiene sus ventajas y desventajas. Una posibilidad que tiene el mismo comportamiento del patrón Singleton seria crear la clase principal con métodos estáticos que encapsule la lógica del juego, sin necesidad de una única instancia global. Otra manera un poco diferente seria usando otro tipo de patrones que creen tantas instancias como sean necesarias como el de Factory Method o Abstract Factory evitando que la instancia sea privada y estática. Su mejora se vería reflejada en la flexibilidad del manejo de configuraciones en los atributos de la clase. Por último, es tambien valido restringir el uso del patrón a los atributos que lo necesiten explícitamente, para evitar complicar el desarrollo del proyecto.