**Patrones de Diseño**

**Proyecto: emoji-java**

Repositorio: https://github.com/vdurmont/emoji-java

**Información general del proyecto:**

El proyecto "emoji-java" es una biblioteca de código abierto en Java que proporciona funcionalidades para trabajar con emojis. Permite la utilización cadenas de texto que contienen emojis y ofrece funciones para buscar, reemplazar y manipular emojis en diferentes formatos.

**Patrón elegido: Singleton**

Singleton es un patrón de diseño de creación, el cual nos permite asegurarnos de que haya una única instancia de una clase, y proporciona el punto de acceso para que podamos acceder a tal instancia desde cualquier parte de nuestro programa.

Como el constructor normal siempre devuelve una clase nueva, gracias al patrón Singleton podremos obtener el objeto ya creado que necesitemos, sin estar creando más y más objetos si no es necesario, o si tal cosa hace parte de una restricción de nuestro programa.

Con singleton:

1. Garantizamos que nuestra clase siempre tenga una única instancia, ya que en ciertos casos no deseamos que se creen más de una vez, como por ejemplo en una base de datos, en un Autenticados (como en el proyecto).
2. Damos un acceso global de tal clase en toda la aplicación, y evitamos que tal instancia se sobreescriba por x o y razón en otra parte de nuestro sistema.

Por lo tanto, para utilizar este patrón debemos de, a la hora de crear nuestra clase, colocar nuestro constructor por defecto como privado, para que no se pueda utilizar más adelante.

También debemos crear un método estático que va a actuar como nuestro “constructor”, este método va a invocar al constructor privado por primera vez, y cuando volvamos a llamarlo, nos va a retornar la instancia ya creada del objeto.

**Estructura general del diseño:**

El proyecto está estructurado en varios paquetes y clases que permiten el manejo de emojis en cadenas de texto. El programa esta estructurado de la siguiente manera:

1. Paquete "com.vdurmont.emoji": Contiene las clases principales del proyecto relacionadas con la gestión y manipulación de emojis.

Emoji: Representa un emoji individual y contiene información sobre su nombre, código y variantes.

EmojiManager: Implementa el patrón Singleton y se encarga de la gestión y acceso a los emojis.

EmojiParser: Proporciona métodos estáticos para analizar y manipular cadenas de texto con emojis.

EmojiCategory: Enumeración que representa las categorías de emojis disponibles.

1. Paquete com.vdurmont.emoji.emoji: Contiene clases relacionadas con la representación y manejo de emojis específicos.

EmojiParser: Procesa y analiza cadenas de texto en busca de emojis.

EmojiWriter: Genera una cadena de texto a partir de un emoji.

A picture containing screenshot, rectangle, text, parallel

Description automatically generated

En el proyecto, el patrón Singleton se utiliza en la clase "EmojiManager" para garantizar que solo haya una instancia de esta clase en todo el sistema y proporcionar un punto de acceso global a esta instancia.

**Información y estructura del fragmento del proyecto donde aparece el patrón**:

Clase EmojiManager:

La clase EmojiManager implementa el patrón Singleton y se encarga de la gestión y acceso a los emojis.

Esta clase tiene un constructor privado, lo que evita que se creen instancias directamente desde otras clases.

La clase define una variable estática y privada llamada INSTANCE del tipo EmojiManager que almacena la única instancia de la clase.

La clase proporciona un método público y estático llamado getInstance()" que se utiliza para obtener la instancia única de EmojiManager. Si no existe una instancia, este método crea una nueva instancia y la asigna a la variable INSTANCE.

Además del método getInstance(), la clase EmojiManager también proporciona otros métodos para acceder y manipular los emojis, como getAll() para obtener todos los emojis y getByUnicode(String unicode) para buscar un emoji por su código Unicode.

Clases y paquetes relacionados:

El paquete com.vdurmont.emoji.emoji contiene clases relacionadas con la representación y manejo de emojis específicos. Estas clases interactúan con la instancia única de EmojiManager para obtener acceso a los emojis y realizar operaciones sobre ellos.

Otras clases en el proyecto que necesitan acceder a los emojis pueden obtener la instancia de EmojiManager utilizando el método estático getInstance(). Esto proporciona un punto de acceso global a la única instancia de EmojiManager en todo el sistema.

La utilización del patrón Singleton en la clase EmojiManager permite tener una única instancia de esta clase en todo el sistema. Esto garantiza que todas las partes del proyecto que necesiten acceder a los emojis utilicen la misma instancia y evita la creación de múltiples instancias innecesarias.

El uso del patrón Singleton en este contexto facilita la gestión y acceso a los emojis, ya que todas las clases y componentes relacionados pueden utilizar la instancia única de EmojiManager de manera consistente y sin conflictos.

**¿Por qué tiene sentido haber utilizado el patrón en ese punto del proyecto? ¿Qué ventajas tiene?**

En el proyecto "emoji-java", tiene sentido utilizar el patrón Singleton en la clase "EmojiManager" por varias razones, lo cual proporciona algunas ventajas:

1. El patrón Singleton garantiza que haya una única instancia de la clase "EmojiManager" en todo el sistema. Esto permite acceder a esta instancia de forma global desde cualquier parte del proyecto. Al utilizar el método estático "getInstance()", se obtiene siempre la misma instancia, lo que facilita la gestión y manipulación de los emojis de manera consistente en todas las clases y componentes que lo requieran.
2. Al restringir la creación de instancias de "EmojiManager" mediante un constructor privado, el patrón Singleton asegura que solo haya una única instancia de esta clase en el sistema. Esto evita la posibilidad de crear múltiples instancias innecesariamente y garantiza que todos los componentes que necesiten trabajar con emojis accedan a la misma instancia, lo que ayuda a mantener la coherencia y evitar inconsistencias en la manipulación de los emojis.
3. Al tener una única instancia de "EmojiManager" en todo el sistema, se reduce la cantidad de recursos utilizados en términos de memoria y procesamiento. Si se crearan múltiples instancias de "EmojiManager", se duplicarían los recursos necesarios para mantener y manipular los emojis, lo cual sería ineficiente. El patrón Singleton permite optimizar el uso de recursos al garantizar que solo haya una instancia y que esta se comparta entre las diferentes partes del proyecto que la necesiten.
4. Al utilizar el patrón Singleton, se facilita la posibilidad de extender o personalizar el comportamiento de "EmojiManager" en el futuro. Al ser una única instancia accesible globalmente, se puede modificar o agregar funcionalidades a "EmojiManager" sin afectar la forma en que otros componentes interactúan con ella. Esto brinda flexibilidad y modularidad al proyecto, permitiendo realizar cambios en la implementación de "EmojiManager" de manera controlada y sin generar impactos no deseados en otras partes del sistema.
5. La utilización del patrón Singleton en la clase "EmojiManager" del proyecto "emoji-java" tiene sentido porque brinda un acceso global y consistente a una única instancia de gestión de emojis, evita la creación múltiple de instancias, ahorra recursos y facilita la extensión y personalización del comportamiento. Estas ventajas contribuyen a un diseño más eficiente, modular y mantenible del proyecto.

**¿Qué desventajas tiene haber utilizado el patrón en ese punto del proyecto?**

El uso del patrón Singleton puede introducir un acoplamiento fuerte entre las clases que dependen de la instancia única de "EmojiManager" y la propia clase "EmojiManager". Esto se debe a que las clases que requieren acceder a "EmojiManager" dependen directamente de su implementación como Singleton. Esto puede dificultar la flexibilidad y la posibilidad de cambiar o sustituir "EmojiManager" en el futuro sin afectar a otras partes de la aplicación.

También puede añadir dificultades para la realización de las pruebas unitarias. Al depender de una única instancia global de "EmojiManager", puede resultar más complicado realizar pruebas unitarias aisladas en otras partes del proyecto. Las pruebas pueden depender del estado y comportamiento específico de "EmojiManager", lo que dificulta la creación de casos de prueba independientes y puede generar interdependencias en el flujo de ejecución de las pruebas.

Este patrón puede agregar cierta complejidad al diseño del proyecto en comparación con otros enfoques. El uso del patrón Singleton implica una estructura y un flujo de control específicos, lo cual puede aumentar la complejidad y la comprensión del código. Además, si se abusa del patrón Singleton en diferentes partes del proyecto, puede resultar en un acoplamiento excesivo y un programa más difícil de mantener y evolucionar.

**¿De qué otras formas se le ocurre que se podrían haber solucionado, en este caso particular, los problemas que resuelve el patrón?**

En lugar de utilizar una única instancia de "EmojiManager", se podrían proporcionar métodos estáticos en la clase misma para acceder y manipular los emojis. Esto permitiría un acceso global a los métodos de "EmojiManager" sin la necesidad de mantener una única instancia.

**Juan Leonardo Rangel Barrera 202215757**