Isabella Caputi-202122075

Taller 5: Patrones

Patron Escogido: Singleton

Proyecto escogido: https://github.com/estsetubal-pa-geral/SingletonLoggerTemplate.git

Información general del patron: El patrón de diseño Singleton se clasifica como un patrón de creación y tiene como objetivo garantizar la existencia de una única instancia de una clase, proporcionando un punto de acceso global a dicha instancia. Para implementar el patrón Singleton de manera efectiva, se requiere que el constructor de la clase Singleton sea privado. La clase Singleton debe contener una instancia estática privada de sí misma, asegurando que solo haya una copia de la instancia en todo el programa.

Información general del proyecto: El proyecto se centra en la implementación de un Logger destinado a registrar acciones relacionadas con un juego en un archivo con tres clases principales: Logger, Gammer y Game.

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

Utilizar el patrón Singleton en la clase Logger se puede utilizar por varias razones. Por un lado, garantiza que solo haya una única instancia de la clase Logger en todo el sistema, evitando conflictos y asegurando coherencia en el registro de acciones. Por otro lado, proporciona un acceso global a la instancia del Logger a través del método getInstance(), facilitando su uso en diferentes partes del código sin necesidad de pasar la instancia como parámetro. También, permite un control centralizado sobre la creación de la instancia del Logger, inicializando los recursos necesarios en el constructor privado de la clase. Evita la creación innecesaria de múltiples instancias, siendo importante en términos de eficiencia y uso de memoria. Por último, proporciona una única entrada para acceder a la instancia del Logger, facilitando la posibilidad de extender o reemplazar el comportamiento del registrador en el futuro sin modificar el código en múltiples lugares.

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

Puede generar un enlace fuerte entre las clases que dependen del Logger y la implementación del Singleton, dificultando la modularidad y flexibilidad del código. También puede complicar las pruebas unitarias al depender de una instancia global. En entornos con múltiples hilos, el Singleton puede generar problemas de concurrencia si se accede frecuentemente a la instancia del Logger. Puede resultar difícil cambiar la implementación del Logger en el futuro debido al mantenimiento de las clases que dependen del Singleton, lo que puede ser costoso y propenso a errores.

¿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 el patrón Singleton, se podría considerar la inyección de dependencias, pasándola como parámetro en el constructor o a través de métodos setter en las clases que la necesiten. Además, se puede explorar otros patrones de diseño específicos para el registro de acciones, como Observer o Strategy pattern, que podrían proporcionar mayor flexibilidad y personalización según los requisitos específicos del proyecto.