Isabella Caputi-202122075

Taller 5: Patrones

Patrón Escogido: Singleton

Proyecto escogido: https://github.com/cd-educ/patronSingleton.git

• Información general del proyecto: para qué sirve, cuál es la estructura general del diseño, qué grandes retos de diseño enfrenta.

El proyecto presentado es un ejemplo aplicado del patrón GoF de Singleton. Este patron restringe la creación de un objeto a una única instancia. Es útil cuando se necesita un objeto exactamente para coordinar las acciones en todo el sistema (Wikimedia, 2023). En este proyecto especifico dada, se crea una clase llamada Logger. Esta clase esta relacionada a otra que se llama App, la cual hace uso de Logger realizar actividades básicas en este sistema, como sumar o restar dos números, para registrar de esta manera el resultado. El reto principal que este diseño enfrenta es garantizar que solo exista una instancia de la clase y proporcionar un acceso global a esta precisa instancia. Esto se logra al declarar los contructores de la clase Logger privados, lo cual evita que esta sea instanciada por otros objetos. Esto proporciona un método estático que devuelve una referencia a la instancia (el método getIstance() que se llama en la clase App y se define en la clase Logger logra esto). Además, también es importante tener en cuenta que, si este patrón se fuera utilizar en exceso, también generaría un acoplamiento fuerte (Cd-Educ, 2020).

• Información y estructura del fragmento del proyecto donde aparece el patrón. Información del patrón aplicado al proyecto: explicar cómo se está utilizando el patrón dentro del proyecto.

El patrón de Singleton se utiliza en el proyecto dado dentro de la clase Logger. Esta se encarga de registrar los resultados de las operaciones realizadas en el App. Esta clase tiene el siguiente código:

```
package Logger;

public class Logger {

   private static Logger instance = null;

   private Logger(){}

   public static Logger getInstance(){

      if(instance == null){

         instance = new Logger();

      }

      return instance;

   }

   public void imprimir(String log){

      /* Si en vez de imprimirlo por consola quisiera guardarlo
```

```
     * por ejemeplo en un archivo, solo tendria que cambiar esta
     * linea y el resto del programa no se enteraria
     * */
     System.out.println(log);
   }
}
```

Como se mencionó anteriormente, el método getInstance() dentro de esta clase es un método estático que devuelve la única instancia de la clase. Por otro lado, el método imprimir() es un método de instancia que se utiliza para imprimir un mensaje. Este método se invoca en la única instancia de la clase Logger. En este proyecto en específico, el patrón Singleton se utiliza para asegurar que solo haya una instancia de la clase Logger, lo que permite un acceso global para registrar los resultados de las operaciones que se realizan en App (Cd-Educ, 2020).

• Información general sobre el patrón: qué patrón es y para qué se usa usualmente.

Singleton es un patrón de diseño que restringe la creación de un objeto a una única instancia. Este patrón es útil cuando se necesita exactamente un objeto para coordinar acciones en un sistema. Les permite a los objetos asegurarse de que tienen solo una instancia, para así proporcionar un acceso fácil a esta y poder controlarla. Este es uno de los patrones más conocidos de GoF, el cual ayuda a solucionar problemas frecuentes en el software orientado a objetos. Este patrón también puede ser utilizado como base para otros patrones de diseño, como lo son patrones de fabrica abstracta, método de fábrica, constructor y prototipo. Un uso común en la vida real de este patrón es para el registro de eventos, porque todos los objetos que buscan registrar mensajes requieren un punto de acceso uniforme y de manera conceptual, escriben a una única fuente. Por lo tanto, el patrón de Singleton proporciona la certeza de que solo exista una instancia de esa clase y normalmente se da un acceso global a esta, declarando los constructores privados y estableciendo un método estático que devuelva una referencia a la única instancia (Wikimedia, 2023).

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

Tiene sentido haber utilizado el patrón en el punto dado del proyecto por diversas razones y ventajas. Por un lado, la clase Logger se utiliza para registrar los resultados de las operaciones realizadas en el sistema. Esto es bastante útil ya que todos los componentes del sistema pueden tener acceso a este registro gracias al patrón Singleton, ya que este provee un punto de acceso global a la instancia de Logger. Por otro lado, como se utiliza este patrón, se asegura que solo haya una instancia de la clase Logger, lo que evita tener problemas de inconsistencias en el registro de los resultados si se fueran a obtener múltiples instancias. Además, permite un control centralizado en la clase de Logger, ya que este controla como y cuando se crea la instancia, reduciendo costos en términos de recursos. Por último, como solo existe una única instancia, los datos registrados van a persistir durante toda la vida del programa, lo cual es bastante útil si se requiere mantener un registro de todo lo que sucede frente a la ejecución de este.

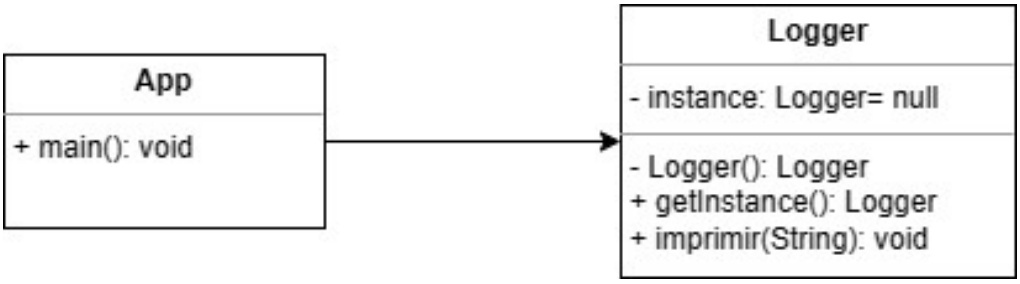
• ¿Qué desventajas tiene haber utilizado el patrón en ese punto del proyecto?

Al tomar la decisión de haber utilizado el patrón en ese punto del proyecto también tiene ciertas desventajas. Por un lado, como se mencionó anteriormente, es posible que haya un acoplamiento fuerte si se utiliza el patrón en exceso. Es decir, los componentes del sistema pueden llegar a depender demasiado de la instancia del Singleton, lo que puede dificultar pruebas del programa y la modificación del código. Adicionalmente, el patrón puede ser difícil de manejar si se intenta crear la instancia de Singleton varias veces al mismo tiempo (por ejemplo, realizar varias operaciones al mismo tiempo), ya que puede haber problemas de sincronización. Por último, es importante tener en cuenta que la instancia del Singleton solo se crea hasta que se necesita. Por lo tanto, puede llegar a ser costoso en cuanto a recursos si la instantacion se fuera a realizar en un momento critico del sistema.

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

En el caso de este proyecto, el patrón Singleton se utiliza para garantizar que solo exista una instancia de la clase Logger y proporciona el acceso global a esta. De todas maneras, existen múltiples formas de abordar estos problemas. Por ejemplo, el patrón de modulo, el cual es un componente de software que se puede importar una sola vez y se reutiliza en todo el programa, de manera similar a un Singleton (Angel, 2023). Por otro lado, el patrón de fabrica se puede utilizar para controlar la creación de objetos, aunque no garantiza una única instancia, puede ser útil para controlar las instancias de una clase (IONOS, 2021). Por último, está la inyección de dependencias, en el cual se pasa una instancia de una clase a las otras clases que requieren de esta, en lugar de tener las clases que crean la instancia directamente (*Qué Es la,* n.d).

Diagrama UML del Proyecto:



Fuente: (Cd-Educ,2020).

Bibliografía

Angel, I. D. (2023). *Patron de Diseño de Módulo, Eso Como se come*. DEV Community. https://dev.to/thelordofth3cloud/patron-de-diseno-de-modulo-eso-como-se-come-5g13

Cd-Educ. (2020). *CD-educ/patronsingleton: Ejemplo Aplicado del Patrón singleton*. GitHub. https://github.com/cd-educ/patronSingleton.git

IONOS. (2021). *Patrón de Diseño Factory: Las Claves del Patrón Factory Method*. IONOS Digital Guide. https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/patron-factory/

Qué Es la Inyección de dependencias y cómo funciona. campusMVP.es. (n.d.). https://www.campusmvp.es/recursos/post/que-es-la-inyeccion-de-dependencias-y-como-funciona.aspx

Wikimedia Foundation. (2023). *Singleton pattern*. Wikipedia. https://en.wikipedia.org/wiki/Singleton_pattern