Taller 5 DPOO

1. **Información general del proyecto, deben incluir la URL para consultar el proyecto.**

   El proyecto consiste en una aplicación con interfaz que muestra un menú con los platos de un restaurante. La aplicación usa un filtrador de comidas en la que se puede agregar o quitar ingredientes de las comidas. En esta aplicación además se usa el patrón singleton para evitar que se repita un objeto específicamente evita que se repita la creación de listas de comidas igualmente evita que posteriormente se creen mas.

   URL: https://github.com/ohm-softa/08-singleton-factory-strategy-android/tree/master/app

2. **Información y estructura del fragmento del proyecto donde aparece el patrón. No se limiten únicamente a los elementos que hacen parte del patrón: para que tenga sentido su uso, probablemente van a tener que incluir elementos cercanos que sirvan para contextualizarlo.**

   En el proyecto que se buscó, implementan el patrón de Singleton en el paquete de (src/main/java/ohm/softa/a08/api), lo que hace es preguntar si ya existe un objeto y hace un llamado a la lista en este la lista de comidas de este proyecto para así, no crear uno nuevo.

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

   Singleton es un patrón de diseño que permite asegurar que una clase sea instanciada una única vez, y también, proporciona un “punto de acceso” para todo el proyecto a partir de esa instancia antes. Singleton ayuda a evitar la multiplicidad en el sentido de creación de objetos por clase. Lo anterior se logra creando el objecto deseado en una clase y llamándolo como una instancia estática. Usualmente, Singleton se usa para controlar el acceso a un único recurso físico o cuando todos los demás objetos de la aplicación deben tener acceso a algún tipo de datos.

4. **Información del patrón aplicado al proyecto: explicar cómo se está utilizando el patrón dentro del proyecto.**

   Singleton se usa para controlar el acceso a un único recurso físico o cuando todos los demás objetos de la aplicación deben tener acceso a algún tipo de datos. En este caso la interfaz de la aplicación está hecha para que puedas eliminar agregar algún ingrediente-comida, realizar esta acción requiere mostrar una lista de

comidas y es acá cuando el patrón singleton es útil ya que en cualquier parte del proyecto podemos ingresar a la lista de ingredientes-comidas sin necesidad de crear un objeto nuevo. En resumen, singleton cumple la función de no repetir el proceso de crear una nueva lista cada vez que se necesite en diferentes partes del proyecto.

En el proyecto tomaban la clase de utilidad para hacerla abstracta, de esta forma, y al declarar en privado el constructor del objeto evitaban la instanciación por parte de otras clases. Sin embargo, esto implicaba que los métodos que debían utilizar debían trabajar en el campo estático. El patrón en este caso era utilizado para que solo existiera un elemento de utilidad y este fuera el que se encargara de todo el proyecto.

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

Usar singleton en este proyecto nos asegura que no se repitan los objetos y genera un acceso a todo el proyecto de este objeto que no queremos que se repita esto gracias a que el objeto en cuestión se instancia una única vez. En este proyecto se usa singleton con el propósito de no tener que crear una interfaz api proxy cada vez que se necesite filtrar las comidas. El uso del patrón tiene ventajas en este proyecto ya que ayuda a que no se tengan que crear muchas implementaciones que devuelvan el mismo resultado en este caso una lista de comidas, con esto el uso de el patrón singleton se permite el acceso a esta función en cualquier parte del proyecto y así se evita tener un programa con ciclos muy grande con diferentes objetos que se usan para crear una lista de comidas.

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

El utilizar Singleton en este punto del proyecto trae algunas desventajas respecto a la división de responsabilidades. Esto quiere decir que el patrón vulnera el *Principio de responsabilidad única* ya que resuelve dos problemas al mismo tiempo; evita el instanciamiento múltiple y genera un punto de acceso global a dicha instancia.

Otra desventaja de utilizar este patrón en el proyecto es que, al ser una utilidad en el código, la clase que lo implementa no puede ser probada con facilidad, el utilizar una clase abstracta dificulta la realización de pruebas unitarias.

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

Otra forma de implementar este patrón en el diseño para solucionar estos problemas seria dejando la clase no abstracta, pero añadiendo un campo estático privado a la misma en donde se almacene la instancia del objeto. Declarar un método de “creación” estático en el cual se obtenga la instancia requerida, de esta

forma, el método deberá devolver siempre esa instancia en todas las llamadas. Por último, declarar el constructor como privado para evitar el instanciamiento, sabiendo que con el método estático se puede devolver la instancia requerida.