Singleton

La mayoría de las recomendaciones indican que es mejor buscar alternativas al patrón de diseño Singleton pues presenta más desventajas que ventajas. Al ser un estado global, genera acoplamiento entre los componentes de la aplicación, lo que a su vez reduce el funcionamiento de las pruebas unitarias. Además, para que funcione adecuadamente, se deben de garantizar varios aspectos de su creación y ciclo de vida. Asimismo, se consideran un “code smell”, pues toma una salida fácil al crear una variable global. Uno de los problemas más serios es que rompe el principio de única responsabilidad, pues también controla su creación. Por lo tanto, aunque reduce el uso de memoria, pues únicamente se crea una instancia, y brinda acceso sencillo a un recursos, se desaconseja su uso. En su lugar, se sugiere implementar estrategias como la inyección de dependencias o el grupo de objetos.

Personalmente nos cuestionamos por qué el patrón de diseño Singleton es tan rechazado en la comunidad y en que situaciones realmente sería aplicable. Para esta hoja de trabajo, consideramos que el uso de un Singleton posiblemente no es el más adeucado. Esto se debe a que la clase que lo implemente es la Calculadora PostFix. Durante la ejecución del programa, el usuario podría necesitar crear más de una calculadora para operar distintos tipos o grupos de números. Además, sería más difícil de mantener si otra clase necesitara usar la calculadora. Por ejemplo, si existe un clase Cajero, posiblemente buscaríamos que cada cajero tuviera su propia calculadora en lugar de todos acceder a la misma. Consideramos que el Singleton tendría una mejor aplicación en otro tipo de clase.