Clase Inmutable

Una clase inmutable es una clase que no se puede modificar después de haber sido creada. Los valores de los objetos creados a partir de la clase no pueden cambiar. Todo lo que se asigne en el momento de la construcción se quedará así durante todo el tiempo que se utilice el objeto.

La inmutabilidad ayuda a evitar errores causados por cambios inesperados en los datos y mejora la seguridad y confiabilidad del código. También hace que sea más fácil trabajar en el código.

Para que una clase sea inmutable se deben seguir varias reglas. Primero es importante usar un constructor para asignar los valores del objeto, lo que quiere decir que toda la información requerida por el objeto se proporcionan desde el principio. Después, todas las variables del objeto deben ser “private” y “final”, lo que significa que no pueden cambiar y tampoco pueden ser vistas fuera de la clase.

Otra regla importante es que no se deben crear “setters” debido a que estos métodos sirven para cambiar valores después de haber creado el objeto. Por lo que si nosotros los creamos se podría modificar el objeto y dejaría de ser inmutable. También hay que tener cuidado con las listas u otros elementos que se puedan modificar.

Otro punto para considerar es que no se debe permitir que otras clases accedan o modifiquen a los objetos. por ejemplo, si el objeto es una lista no se debe regresar ese mismo objeto en un método. En lugar de regresar el mismo objeto se debe regresar una copia, para evitar que lo cambien por fuera y afecten el contenido original.

Además, se debe marcar a la clase como “final” para que no se pueda heredar o modificar su comportamiento. Así se asegura que la clase siempre funcione como fue planeado y no se le pueda hacer cambios por medio de una subclase.

Un ejemplo muy conocido de una clase inmutable en Java es la clase String, la cual no permite modificar su contenido una vez creado el objeto. Esto facilita su uso en aplicaciones donde la seguridad, la estabilidad y la integridad de los datos son requeridas.

Singleton

El patrón Singleton se usa en Java cuando queremos asegurarnos de que una clase tenga una sola instancia en toda la aplicación. Con esto se refiere a que no importa cuántas veces llamemos a esa clase, siempre se utilizará el mismo objeto y no se crearán copias nuevas.

El Singleton se utiliza en situaciones donde si se tuvieran varios objetos habría problemas o confusión. Por ejemplo, si tenemos una clase que se encarga de guardar la configuración general de un programa, lo más adecuado es que existiera solo una instancia para que no haya diferencias o errores en los datos. También se utiliza mucho en las conexiones de las bases de datos o en clases que manipulan la seguridad del sistema.

Para que el patrón funcione adecuadamente es necesario seguir ciertas reglas. Primero, no se permite crear objetos de la clase desde fuera de ella. Esto se hace ocultando la forma en que normalmente se crean los objetos. Después, se hace que la clase solo tena una única forma de acceder al objeto y siempre se devuelve el mismo objeto. Si todavía no existe el objeto hay que crearlo pero si ya existe se usa el mismo que ya se había creado anteriormente.

Lo bueno de utilizar el patrón Singleton es que ahorra memoria y mejora el rendimiento de los sistemas porque no se están creando muchos objetos que hagan lo mismo. Además, como se tiene solo una instancia, es más fácil controlar su comportamiento debido a que todos usan el mismo recurso compartido.

A pesar de esto, es necesario tener cuidado cuando se usa en aplicaciones que funcionan con varios hilos al mismo tiempo, porque podrían intentar crear el objeto más de una vez al mismo tiempo pero por eso existen formas más seguras de aplicar el patrón, pero la idea principal sigue siendo que solo haya un único objeto.

Entre las formas seguras de aplicar el patrón se encuentra el uso de la palabra “synchronized” para que solo un hilo pueda crear la instancia. Otra forma es el bloqueo doble el cual revisa dos veces si ya existe el objeto antes de crearlo. También se puede usar una clase interna que se cargue solo cuando se necesita lo que lo hace ser más rápido y seguro. Por último, se puede usar un tipo llamado “enum” la cual es la forma más segura porque Java se encarga de todo automáticamente.