**Ejercicios sesiones 6, 7 y 8**

Identifica las características principales de los patrones de diseño más conocidos, indica para qué valen y un caso de uso para cada uno de ellos.

* **Factory method:** Proporciona una interfaz para la creación de objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán. Define un método que debe utilizarse para crear objetos, en lugar de una llamada directa a un constructor. Se utiliza cuando se quiere ahorrar recursos del sistema mediante la reutilización de objetos existentes en lugar de reconstruirlos cada vez.
* **Singleton:** Permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia. Lo utilizan para restringir la creación de objetos de una clase.
* **Observer:** Permite definir un mecanismo de suscripción para notificar a varios objetos sobre cualquier evento que le suceda al objeto que están observando. Proporciona una forma de reaccionar a los eventos que suceden en otros objetos, sin acoplarse a sus clases. Permite a un objeto notificar a otros objetos sobre cambios en su estado.
* **Strategy:** Permite definir una familia de algoritmos, colocar cada uno de ellos en una clase separada y hacer sus objetos intercambiables. Es un patrón de diseño de comportamiento que convierte un grupo de comportamientos en objetos y los hace intercambiables dentro del objeto de contexto original. Se utiliza como una forma de cambiar el comportamiento de una clase sin extenderla.
* **Adapter**: Permite la colaboración entre objetos con interfaces incompatibles. Se utiliza muy a menudo en sistemas basados en algún código heredado (*legacy*). En estos casos, los adaptadores crean código heredado con clases modernas. Se utiliza cuando se desea **usar** una clase existente, y su interfaz no sea igual a la necesitada. O cuando se desea crear una clase reutilizable que coopere con clases no relacionadas. Es decir, que las clases no tienen necesariamente interfaces compatibles.
* **Builder**: es un patrón de diseño creacional que permite construir objetos complejos paso a paso.. Este patrón nos permite producir distintos tipos y representaciones de un objeto empleando el mismo código de construcción. Resulta especialmente útil cuando debes crear un objeto con muchas opciones posibles de configuración.
* **State:** es un patrón de diseño de comportamiento que permite a un objeto cambiar de comportamiento cuando cambia su estado interno. Se utiliza habitualmente en Java para convertir las enormes máquinas de estados basadas en declaraciones switch, en objetos.

Crea una pequeña aplicación implementando el patrón Singlenton y demuestra que, efectivamente, no crea instancias nuevas. Puedes tomar la sesión 7 como referencia.





