Patrones de Diseño https://github.com/alxgcrz/_design-patterns_

1. **Singleton**: Este patrón es crucial en escenarios donde se necesita una única instancia de una clase a lo largo de la aplicación.

2. **Factory**: Este patrón se destaca por su capacidad de encapsular la creación de objetos, permitiendo la flexibilidad en la elección del tipo de objeto a crear.

3. **Observer**: Este patrón permite que un objeto notifique a otros objetos cuando cambia su estado.

4. **Decorator**: Este patrón permite agregar comportamiento adicional a un objeto de manera dinámica.

5. **Strategy**: Este patrón permite seleccionar un algoritmo en tiempo de ejecución.

6. **Composite**: Este patrón permite trabajar con objetos individuales y composiciones de objetos de manera uniforme.

7. **Adapter**: Este patrón permite que las interfaces de una clase existente se utilicen como otra interfaz.

8. **Prototype**: Este patrón permite copiar objetos existentes sin hacer que el código dependa de sus clases.

9. **Builder**: Este patrón permite construir objetos complejos paso a paso.

10. **Facade**: Este patrón proporciona una interfaz simplificada a una biblioteca, un marco o cualquier otro conjunto de clases complejas.

11. **Bridge**: Este patrón divide una clase grande o un conjunto de clases estrechamente acopladas en dos jerarquías separadas.

12. **Command**: Este patrón convierte una solicitud en un objeto independiente que contiene toda la información sobre la solicitud[3].

13. **Chain of Responsibility**: Este patrón permite pasar solicitudes a lo largo de la cadena de manejadores.

14. **Mediator**: Este patrón reduce las dependencias entre los objetos comunicantes.

15. **Flyweight**: Este patrón permite ajustar más objetos en la cantidad disponible de RAM al compartir partes comunes de estado entre múltiples objetos.

16. **State**: Este patrón permite que un objeto altere su comportamiento cuando su estado interno cambia.

17. **Proxy**: Este patrón proporciona un sustituto o marcador de posición para otro objeto.

18. **Iterator**: Este patrón permite a los clientes recorrer los elementos de una colección sin exponer su representación subyacente.

19. **Memento**: Este patrón permite guardar y restaurar el estado previo de un objeto sin revelar los detalles de su implementación.

20. **Visitor**: Este patrón permite agregar operaciones a objetos sin cambiar las clases de los objetos en los que operan.