Patrones de diseño propuestos

1. Patrón Singleton

Descripción: El patrón Singleton asegura que una clase tenga solo una instancia y proporciona un punto de acceso global a esa instancia. Esto es útil para gestionar recursos compartidos, como configuraciones globales.

- **Ventajas:**
  - **Control Centralizado:** Garantiza un único punto de acceso para configuraciones o recursos globales, evitando inconsistencias.
  - **Uso Eficiente de Recursos:** Minimiza el uso de memoria al asegurar que solo exista una instancia.
- **Utilidad en el Proyecto:** Se puede utilizar el patrón Singleton para la configuración global de la plataforma, como la autenticación del administrador y la gestión de reportes, asegurando que solo haya una instancia que controle estas operaciones críticas.

2. Patrón Factory Method

Descripción: El patrón Factory Method define una interfaz para crear objetos, pero permite a las subclases decidir qué clase instanciar. Es útil cuando el tipo de objeto a crear no se conoce hasta el momento de ejecución.

- **Ventajas:**
  - **Flexibilidad:** Permite la creación de objetos sin especificar la clase exacta, facilitando la adición de nuevos tipos de usuarios o roles.
  - **Desacoplamiento:** Reduce el acoplamiento entre el código cliente y las clases concretas que se crean.
- **Utilidad en el Proyecto:** Se puede usar este patrón para crear diferentes tipos de usuarios (Usuario, Conductor, Asistente) sin que el código de la aplicación tenga que conocer las clases específicas que se instancian.

3. Patrón Observer

Descripción: El patrón Observer define una dependencia uno a muchos entre objetos, de manera que cuando un objeto cambia de estado, todos sus dependientes son notificados y actualizados automáticamente.

- **Ventajas:**
  - **Desacoplamiento:** Permite que el sujeto y los observadores estén desacoplados, facilitando la expansión y mantenimiento del sistema.
  - **Notificaciones Automáticas:** Proporciona un mecanismo eficiente para notificar a múltiples objetos sobre cambios en el estado.
- **Utilidad en el Proyecto:** Se puede utilizar el patrón Observer para notificar a los conductores sobre nuevas solicitudes de viaje, asegurando que solo un conductor acepte cada viaje y evitando conflictos.

4. Patrón Strategy

Descripción: El patrón Strategy define una familia de algoritmos, encapsula cada uno y los hace intercambiables. Permite que el algoritmo varie independientemente de los clientes que lo utilizan.

- **Ventajas:**
  - **Intercambiabilidad de Algoritmos:** Permite cambiar el comportamiento del sistema de forma dinámica.
  - **Separación de Responsabilidades:** Facilita la modificación o adición de nuevos algoritmos sin alterar el código cliente.
- **Utilidad en el Proyecto:** Este patrón es útil para calcular tarifas de viaje basadas en zonas, permitiendo la adición o modificación de estrategias de cálculo sin cambiar el código del sistema principal.

5. Patrón Command

Descripción: El patrón Command encapsula una solicitud como un objeto, así que puedes parametrizar los objetos con diferentes solicitudes, colas o registros de solicitudes, y deshacer operaciones.

- **Ventajas:**
  - **Desacoplamiento de Solicitudes:** Separa el objeto que invoca la operación del que sabe cómo realizarla.
  - **Deshacer y Rehacer:** Permite implementar operaciones de deshacer y rehacer fácilmente.
- **Utilidad en el Proyecto:** puede ser útil para manejar operaciones como aceptar y cancelar viajes, permitiendo una gestión flexible de estos comandos y facilitando la implementación de características adicionales como deshacer una acción.

6. Patrón Decorator

Descripción: El patrón Decorator permite añadir responsabilidades adicionales a un objeto de forma dinámica. Proporciona una alternativa flexible a la sub-clasificación para extender la funcionalidad de los objetos.

- **Ventajas:**
  - **Extensibilidad:** Permite añadir funcionalidades a los objetos de forma dinámica sin modificar su código.
  - **Flexibilidad:** Los objetos pueden ser decorados con múltiples funcionalidades de manera combinada.
- **Utilidad en el Proyecto:** se puede aplicar el patrón Decorator para añadir funcionalidades adicionales a los usuarios y conductores, como reportar problemas o agregar calificaciones, sin alterar las clases originales.