Taller 5

Hotel-Management-Project-Java:
https://github.com/shouryaj98/Hotel-Management-Project-Java.git

El proyecto que yo escogí está programado en Java y es sistema de gestión de un hotel como su título indica. A continuación indico algunas funcionalidades clave que se pueden identificar:

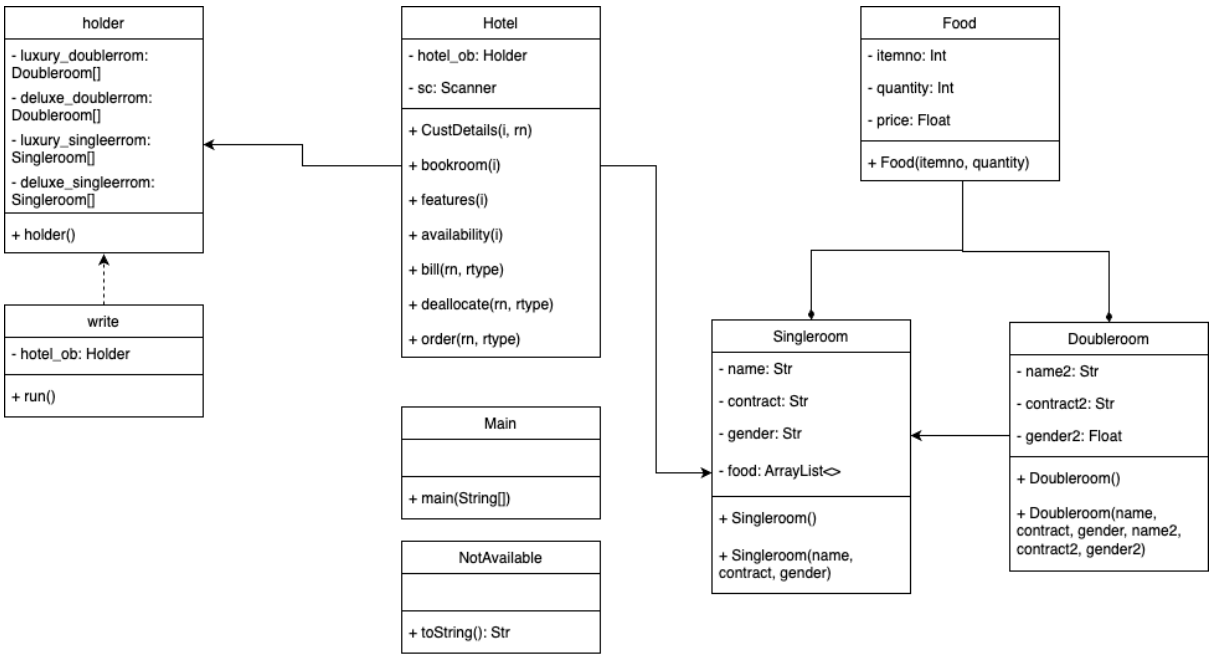
- **Reserva de Habitaciones:**
  Permite a los usuarios reservar diferentes tipos de habitaciones, como habitaciones dobles de lujo, habitaciones dobles de lujo, habitaciones individuales de lujo y habitaciones individuales de lujo.

- **Gestión de Clientes:**
  Registra detalles sobre los clientes que reservan habitaciones, incluyendo nombre, contacto y género.

- **Gestión de Alimentos:**
  Permite a los clientes realizar pedidos de alimentos, especificando el tipo de alimento y la cantidad.

- **Facturación:**
  Calcula la factura total para los clientes, que incluye el costo de la habitación y los alimentos pedidos.

- **Persistencia de Datos:**
  Utiliza serialización para guardar y cargar datos relacionados con las habitaciones y reservas.



- **Clases de Habitaciones:**
  "*Singleroom*" y "*Doubleroom*" representan habitaciones individuales y dobles, respectivamente. Ambas clases heredan de Serializable y contienen detalles sobre el nombre del cliente, contacto y género.

- **Clase Food:**
  Representa los alimentos pedidos por los clientes y almacena detalles como el número de artículo, cantidad y precio.

- **Clase NotAvailable:**
  Es una excepción personalizada que se lanza cuando se intenta reservar una habitación ya ocupada.

- **Clase holder:**
  Contiene arreglos de habitaciones individuales y dobles para mantener el estado del hotel.

- **Clase Hotel:**
  Contiene métodos para la gestión de habitaciones, reservas, alimentos, facturación y persistencia de datos. También utiliza la clase write para la escritura asíncrona de datos en un archivo de respaldo.

- **Clase write:**
  Implementa la interfaz Runnable y se utiliza para describir el estado del hotel de manera asíncrona en un archivo de respaldo.

Algunos de los desafíos o aspectos difíciles del diseño podrían incluir:

I. **Manejo de Excepciones:**
   La gestión de excepciones, como la clase "*NotAvailable*", es importante para manejar situaciones en las que una habitación ya está reservada. Sin embargo, el manejo de excepciones puede volverse complejo a medida que el sistema crece.

II. **Gestión de Datos:**
   La gestión de datos de habitaciones, clientes, alimentos, y la persistencia de estos datos a través de la serialización puede ser complicada y propensa a errores si no se maneja adecuadamente.

III. **Interacción Asíncrona:**
   La utilización de un hilo (write) para la escritura asíncrona de datos introduce concurrencia en el sistema. El diseño debe ser cuidadoso para evitar problemas de concurrencia y garantizar la consistencia de los datos.

IV. **Complejidad del Menú y Pedidos:**
La implementación actual de la orden de alimentos se realiza en el método order, donde el cliente puede hacer múltiples pedidos. La gestión de menús y pedidos más complejos podría requerir una estructura más robusta.

V. **Persistencia y Recuperación de Datos:**
La persistencia de datos mediante serialización puede tener limitaciones en términos de flexibilidad y escalabilidad. Podría ser beneficioso considerar opciones más avanzadas de persistencia y recuperación de datos.

Singleton:

El patrón que voy a analizar dentro de este código es el patrón Singleton, el patrón Singleton nos permite asegurarnos de que una clase tenga una única instancia y proporciona un punto de acceso global a dicha instancia.
Normalmente este patrón es usado en las clases de log in, administrar conexiones a bases de datos y administrar archivos.

Singleton dentro del proyecto:

A. **¿Cómo se está utilizando el patrón dentro del proyecto?**
La clase "*Hotel*" tiene una variable estática "*hotel_ob*", que parece representar una única instancia de la clase "*holder*". Sin embargo, no hay un constructor privado y un método estático para obtener la instancia única

B. **¿Por qué tiene sentido haber utilizado el patrón en ese punto del proyecto? ¿Qué ventajas tiene?**
Garantiza que exista solo una instancia de la clase "*Hotel*" en todo el sistema, lo que podría ser útil para centralizar el estado del hotel y compartirlo entre diferentes componentes del sistema.

C. **¿Qué desventajas tiene haber utilizado el patrón en ese punto del proyecto?**
Puede haber problemas si no se maneja correctamente en un entorno multiproceso o si se desea permitir la extensión de múltiples instancias en el futuro.

D. **¿De qué otras formas se le ocurre que se podrían haber solucionado, en este caso particular, los problemas que resuelve el patrón?**
En lugar de un Singleton, se podría considerar el uso de una clase de gestión centralizada que contenga el estado del hotel y se inyecte en las clases que lo necesiten.