Universidad Nacional de Quilmes

ooo

POO2

Trabajo Final

Sistemas de Alquileres temporales

Integrantes:

- Chiappori Valentina. valchiappori@gmail.com
- Colón Lucrecia Milagros. lucrecolon1828@gmail.com
- Tórboli Nicolás. nicolastorboli1@gmail.com

Decisiones de diseño:

- Las listas de reservasConfirmadas (Inmueble) y reservas(Usuario) almacena tanto las reservas aceptadas como las finalizadas.
- Asumimos que el estado inicial por defecto de una Reserva será “Pendiente”.
- La lista de reservasPendientes se encarga de almacenar aquellas que están a la espera de ser rechazadas o aceptadas por el propietario del inmueble. Mientras que la lista de reservasEnCola solo guarda aquellas reservas que se hayan enviado una vez que el inmueble se encuentre ocupado.Si la reserva original es cancelada, la primera de la cola pasa a reservasPendientes, repitiendo el ciclo normal del envío de una reserva.
- Incluimos un MailSender para el sitio, con el fin de que se encargue del envío de mails.
- Asumimos que realizar el checkOut siempre será válido mientras se haga en el mismo día de salida o cualquier día después, porque entendemos que el checkout en el sitio no necesariamente está relacionado con la salida del inmueble de los inquilinos en la vida real.
- Asumimos que el precio de un Inmueble nunca será un número negativo.
- Asumimos que para obtener todos los inmuebles libres la búsqueda/filtrado se realiza en aquellos disponibles hoy.
- Asumimos que un Propietario puede visualizar o no visualizar a un inquilino antes de aceptar o rechazar una reserva.

Detalles de implementación:

- El método enviarMailConReserva(Reserva reserva) queda a disposición para ser utilizado en el futuro, ya que no está especificado en qué momento debe realizarse la notificación.
- Se decidió utilizar **enums** para manejar los eventos y devolver un mensaje según el tipo de evento, lo cual aporta mayor claridad y simplicidad al representar los eventos de manera explícita, además de mejorar la escalabilidad y seguridad al garantizar que los valores de los eventos sean correctos. Sin embargo, la desventaja principal es que los mensajes asociados a los eventos son estáticos y fijos, lo que significa que cualquier cambio en el mensaje requeriría modificar el código o crear nuevos métodos, limitando la flexibilidad en la personalización de los mensajes.

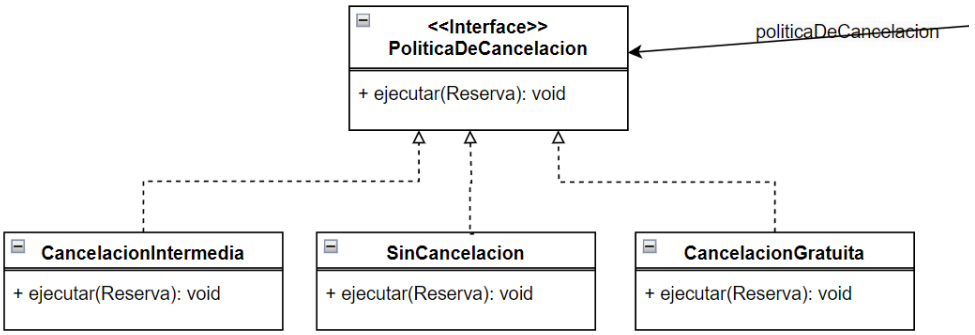
En este proyecto, se utilizaron los patrones de diseño **Observer** y **Strategy**, los cuales se detallarán a continuación, explicando sus respectivos roles.

Strategy:

En este caso, el patrón se aplica para definir diferentes políticas de cancelación que pueden variar según las condiciones. Cada política de cancelación es una estrategia independiente que se implementa como una clase concreta que define cómo calcular el cargo por cancelación en función de la fecha. La interfaz PoliticaDeCancelacion define el comportamiento común para todas las políticas, mientras que las clases concretas implementan las reglas específicas de cada política.

Roles:

- **Context:** Inmueble (el objeto que utiliza la estrategia de cancelación).
- **Strategy:** PoliticaDeCancelacion.
- **ConcreteStrategy:** CancelacionGratuita, SinCancelacion y CancelacionIntermedia.



Observer:

En este caso, Inmueble actúa como el Subject y notifica a los objetos registrados (los Observers) cuando ocurre un evento como una baja de precio o cancelación de reserva. Los Observers concretos, como SitioObservador y AplicacionMobile, implementan la interfaz Interesado y realizan acciones específicas al recibir la notificación, como mostrar mensajes en la página web o en la aplicación móvil. Este enfoque permite añadir nuevos interesados sin modificar el código existente, manteniendo el sistema flexible y escalable.

Roles:

- **Subject:** Inmueble
- **Observer:** Interesado
- **ConcreteObserver:** SitioObservador y AplicacionMobile