Universidad Nacional de Quilmes

Ciencia y Tecnología

Carrera/s: Tecnicatura en Programación Informática / Licenciatura en Informática

Asignatura:

Programación Orientada a Objetos II

Primer cuatrimestre de 2021

Fecha: 15 de julio de 2021

Docente: Agüero, Axel

Tarea: Trabajo Final. Sistema de Alquileres temporales

Alumnos: Gonzalez, Agustín
Greco Ventura, Germán
Espinoza, Braian

Mail de contacto: agustin-ag@live.com
germangv1998@gmail.com
braian3000gmail.com

INDICE

DISEÑO

Dado que el Sistema de Alquileres temporales debe tener diferentes responsabilidades decidimos optar por separar cada responsabilidad en clases distintas pudiendo respetar los principios de SOLID, siendo en este caso el Single responsibility principle.

A su vez, tuvimos que crear distintos objetos abstractos como pueden ser las Políticas de Cancelación, los Estados de las Reservas o Temporada permitiéndonos garantizar el principio de Open/closed principle de forma tal que al haber trabajado de manera polimórfica nos aseguramos que en el momento que el sistema necesite incorporar nueva información pueda realizarse sin estar modificando o borrando lo ya creado.

Un dato que se aprecia en el diseño es que el Administrador posee los diferentes Tipos de Inmuebles, servicios o características tanto de ellos como de los usuarios registrados en el sitio; por lo que al momento de rankear a un elemento del sistema, la publicación de un inmueble o la realización de una reserva primero se verifica que tanto el Usuario como el Inmueble cumplan con los requisitos para hacerlos.

Dentro de estos requisitos se encuentran, por ejemplo, los Estados de las Reservas permitiendo realizar al Usuario diferentes funciones, siendo realizar el CheckOut solamente cuando ya haya concluido con el hospedaje.

Una novedad que tiene el sitio es que diferentes entidades pueden suscribirse a un inmueble para recibir notificaciones dependiendo del tipo de suscripción que tengan, como son la baja de precio, la cancelación o la reserva del mismo. Para ello, se implemento un sistema en el al momento de realizar algunos de los eventos ya nombrados se enviarán las notificaciones solamente a aquellos que les corresponda, evitando la sobreinformación para aquellos que no se encuentren interesados.

A continuación, se adjuntará los enlaces para poder acceder al repositorio y al UML con el diseño completo.

Repositorio GitHub: https://github.com/germangrecoventura/unqui-po2-TpFinal
Diseño UML: https://github.com/germangrecoventura/unqui-po2-TpFinal/blob/main/UML_SistemaDeAlquileresTemporales.png

PATRONES

Patrón Strategy

Al analizar las consignas y el UML, observamos que el Inmueble debía tener 2 objetos (Temporada y PoliticaDeCancelación) que, según como se establecieran, debían comportarse de manera diferente.

Decidimos utilizar Strategy ya que permite que el usuario pueda configurar los distintos tipos de precio (Temporada) y las políticas de cancelaciones. Es decir, elegir una de las posibles formas que ambas pueden tomar, por ejemplo, un precio TemporadaAlta o FinesDeSemanaLargo,etc., mientras que, por el lado de las políticas de cancelaciones, el poder tener la posibilidad de elegir una cancelación gratuita, sin cancelación o intermedia.

Otro motivo por el que decidimos utilizar Strategy es porque permite cambiarse en medio de su ejecución sin generar ningún inconveniente. Gracias al polimorfismo que posee el patrón deja abierta la programación para la incorporación de nuevos métodos.

Roles para los distintos precios:

- Context -> Inmueble
- Strategy -> Temporada
- ConcreteStrategyA-> FeriadosInamovibles
- ConcreteStrategyB-> TemporadaAlta
- ConcreteStrategyC-> FinesDeSemanaLargo
- ConcreteStrategyD-> Fijo

Roles para las distintas PolíticasDeCancelaciones:

- Context -> Inmueble
- Strategy -> PoliticaDeCancelacion
- ConcreteStrategyA-> CancelaciónGratuita
- ConcreteStrategyB-> SinCancelación
- ConcreteStrategyC-> CancelaciónIntermedia

Patrón State

Lo utilizamos para modelar los estados de la reserva. Esto se debe a que, al estar dependiendo del estado en que se encuentre, pueda realizar distintas acciones, como por ejemplo puntuar o comentar a los usuarios y a los inmuebles bajo una condición en particular, como es el caso de estar en CheckOut.

La manera en la que fuimos cambiando los estados de la misma, fueron a través de mensajes de cambio, como, por ejemplo: cancelar, aceptar o finalizar. Cada estado

cambia a estados distintos de la reserva. Por ejemplo cuando se quiere puntuar a alguien, y el estado de la reserva no lo permite (estadoPendienteDeAprobacion, por ejemplo) retorna un error. Ya que solo se puede puntuar o comentar, cuando el estado de la reserva es finalizado.

Roles

- Context -> Reserva
- State -> EstadoDeReserva
- ConcreteStateA -> EstadoReservaAprobado
- ConcreteStateB -> EstadoReservaFinalizado
- ConcreteStateC -> EstadoPendienteDeAprobacion
- ConcreteStateD -> EstadoReservaCancelado
- ConcreteStateE -> EstadoReservaCondicional
- ConcreteStateF -> EstadoReservaConcretado

Patrón Observer

Decidimos utilizar el patrón observer ya que nos permite notificar a los suscriptores de alguno o todos los eventos del inmueble, sobre los cambios de en los eventos que este posea. En este caso en particular el Inmueble cuenta con 3 eventos:

1. Baja de precio de un Inmueble
2. Cancelación de un Inmueble
3. Reserva de un Inmueble

Al realizarse alguno de ellos, el sistema filtra mediante una característica del evento ocurrido a quienes deben de serles notificados, permitiendo optimizar el sistema y dejando flexibilidad a la hora de agregar nuevos tipos de eventos.

Roles para las notificaciones

- Subject -> Observable
- ConcreteSubject-> Inmueble
- Observer> IObserver
- ConcreteObserverA-> AplicacionMobile
- ConcreteObserverB-> Trivago

Patrón Composite

Inicialmente para poder realizar las búsquedas de inmuebles según los filtros correspondientes utilizamos ifs anidados. El problema de esto es que si se desea agregar más filtros se debe modificar lo ya hecho por lo que se violaría el SOLID por el Open-Closed. Para solucionarlo, utilizamos el patrón Composite que nos permite construir objetos complejos mediante objetos más sencillos de forma recursiva. Otro punto positivo es que es facil añadir o remover componentes, en este caso, los fitros.

En este caso, el Composite contendrá todos los filtros y de forma recursiva devolverá el resultado sin preocuparnos por los tipos de los mismos.

Roles para los filtros

- Client -> Sitio
- Component -> IFiltroDeBúsqueda
- Composite -> FiltroComposite
- Leaf -> FiltroCiudad
- Leaf -> FiltroFechaEntradaYSalida
- Leaf -> FiltroCapacidadDeHuéspedes
- Leaf -> FiltroPrecioMínimoYMáximo