Universidad Nacional de Quilmes

PROGRAMACIÓN CON OBJETOS II

TRABAJO PRÁCTICO FINAL

SISTEMA DE ALQUILERES TEMPORALES

PROFESORES:

- DIEGO TORRES
- DIEGO CANO
- MATIAS BUTTI

ALUMNOS:

- JULIÁN VÁZQUEZ // *julivazquez09@gmail.com*
- ALAN PACHECO // *pachecoalannahuel@gmail.com*
- DAVID CARREVEDO // *davidcarrevedo@gmail.com*

https://github.com/Nahuel-P/unqui-po2-alquileres-temporales

INDICE

https://github.com/Nahuel-P/unqui-po2-alquileres-temporales

ALCANCE

DESCRIPCION DEL SISTEMA

Se implementa la lógica de un Sitio Web con la capacidad de comunicar Usuarios que poseen propiedades en alquiler, y Usuarios interesados en alquilarlas.

Se desarrolla en lenguaje Java, con cobertura de Tests JUnit 5 mayor al 95% y el uso de Mockito como asistencia en los Tests.

DECISIONES DE DISEÑO

WEB RESERVAS

Se pensó en este objeto como dispositivo de entrada al sistema, con la idea de manejar las funcionalidades más generales del programa e influyendo en varias de las otras clases, pero sin perder de vista el Single Responsability. Es por esto que las funcionalidades complejas fueron delegadas a otras clases.

BUSQUEDA MEDIANTE FILTRADO

Luego de haber intentado implementar 2 patrones de diseño, el filtrado terminó no coincidiendo exactamente con ningún patrón.

El filtrado se hará partiendo de una interfaz Filtro, la cual quedará abierta a recibir nuevos criterios de búsqueda cualesquiera sean. Cada criterio será una nueva clase que implementará la lógica de ese filtrado, y deberá saber reponder si una Publicación cumple con su criterio debido a que implementa la interfaz Filtro.

En este caso se implementan el filtrado Básico, filtrado por Precio, y filtrado por Cantidad de Habitantes.

USUARIO

También susfrió modificaciones en su idea original. En un principio fue pensado como una clase abstracta, que tendría clases hijas que definirían el comportamiento del usuario, luego se llegó a la conclusión de que una misma instancia de usuario podría querer tener distintos tipos de funcionalidades y que no se lo podía limitar, por eso se mantuvo como clase concreta y única.

Entre sus funciones, se destacan las de crear y recibir reservas/publicaciones

https://github.com/Nahuel-P/unqui-po2-alquileres-temporales

PUBLICACION

Una publicación será la encargada de combinar información de distintas clases en un lugar. Guardará un único Inmueble y podrá acceder a todos sus datos; y a través suyo, un Usuario también podrá acceder a datos de Inmuebles.

Conoce a una clase Observer y posee un método de notificación, de esta manera podra avisar a distintos observadores dependiendo sus intereses *(ver OBSERVADORES DE RESERVAS Y PUBLICACIONES)*.

Una publicación, además, podrá tener diferentes precios dependiendo la fecha en la que sea reservada *(ver TEMPORADAS ESPECIALES)*

RESERVA

Es la concreción de la interacción entre 2 Usuarios.

Al identificar que una misma instancia de Reserva puede tener distintos estados e ir cambiando en tiempo de ejecución, se resuelve usar el patrón State para tratar cada uno de los estados en un momento dado.

De esta manera, también se deja abierta la posibilidad de seguir agregando futuros estados, respetando así el principio SOLID de Open/Closed.

BIBLIOTECA DE RESERVAS Y PUBLICACIONES

Se basan en la lógica del patrón de diseño Facade.

El objetivo es concentrar en distintas bibliotecas (fachadas) las distintas reservas y publicaciones que se vayan concretando. De esta manera logramos quitar a la web la responsabilidad del guardado, dándole independencia y desacoplandola de los clientes que necesiten dicha información. Incluso se vuelve reutilizable con clases futuras que pudieran requerirla.

OBSERVADORES DE RESERVAS Y PUBLICACIONES

En ambos casos, los observadores tendrán la posibilidad de suscribirse a la clase de la cual deseen recibir notificaciones. También, ambas clases lo hacen a través de una interfaz, dejando abierta la posibilidad de recibir distintos tipos de observadores y respetando Open/Closed. La decisión de mantener una clase Observer, encargada de agrupar una lista de observadores y sus respectivos métodos de gestión dentro de la case, tiene como objetivo quitarle responsabilidades al sujeto observado, en este caso la publicación.

En el caso puntual de las reservas, el observador será con listeners, haciendo que nuestra interfaz sea un poco más compleja. Esto choca contra el principio de Segregación de Interfaces, pero aporta la correcta funcionalidad que se buscaba.

CALIFICACIONES

Tanto Usuarios como Inmuebles son calificables. Esto hizo pensar en que debería existir una interfaz que nos permitiera tratarlos polimórficamente. Luego se pudo ver que ninguna clase utilizaría ese polimorfismo, por lo tanto, no era necesaria y se eliminó.

Las calificaciones serán guardadas en una Biblioteca de Calificaciones, volviendo a uitilizar la lógica de Facade, dándole independencia al Sitio Web y haciendo reutilizables los datos. Este objeto tiene la finalidad de gestionar las calificaciones que se le dan a un usuario o inmueble, permitiendo no solo acceder a los comentarios y puntajes, sino que también, entregando un promedio de puntuaciones en caso de que un usuario del sistema quiera conocer esta información de otro.

TEMPORADAS ESPECIALES

Surgen de la necesidad de tener una forma de crear distintos precios para distintas fechas, ya que un Usuario puede determinar varias temporadas con precios de descuento.

Es así que la Publicación delegará la tarea de establecer estos precios a la clase Precio Temporal, y ella (la Publicación) se encargará únicamente de almacenarlos.

POLITICAS DE CANCELACON

Cada Publicación se crea con una política de cancelación determinada, por lo tanto, cuando se cancela una reserva que se hizo sobre una publicación, el precio a cobrar por dicha reserva será determinado por la política de cancelación con al que fue creada dicha reserva.

El diseño de las distintas políticas de cancelación fue sujeto a debate en reiteradas ocasiones. Primero se cambió el diseño original, que utilizaba condicionales que evaluaban los días restantes hasta la entrada en vigencia de la reserva para calcular el monto a indemnizar, por un diseño que utiliza un objeto por cada porcentaje a cobrar en las políticas de cancelación intermedia. El problema con este diseño fue, que seguía evaluando condiciones y no se volvía realmente escalable porque seguía analizando condiciones y en caso de agregar nuevos porcentajes de cobro, no encontramos forma de que respete el principio open-close. Por lo tanto, se decidió sacrificar lo evolutivo a cambio de no caer en un sobre diseño.

En una última instancia se había pensado en una lista de porcentajes a indemnizar, cada una con un método de análisis para conocer los días restantes y si era posible aplicar. De manera tal que en función de la cantidad de días restantes, se recorra la lista y donde hubiese coincidencia entre estos días y la condición impuesta por la política, se asuma como porcentaje el provisto por la clase. Sin embargo, por cuestiones de tiempo, no fue implementado.

https://github.com/Nahuel-P/unqui-po2-alquileres-temporales

(continúa abajo)

PATRONES DE DISEÑO UTILIZADOS

OBSERVER

- Publicación / IPriceObserver
  - ConcreteSubject: Publicación
  - Subject: Observer
  - Observer: IPriceObserver
  - ConcreteObserver: clase que implemente la interfaz IPriceObserver

OBSERVER CON LISTENER

- Reserva / IBookingListener
  - ConcreteSubject: Reserva
  - Subject: Reserva
  - Observer: BookingListener
  - ConcreteObserver: clase que implemente la interfaz IBookingListener

FACADE

- WebReservas / BibliotecaDeReservas / Reserva
  - Client: WebReservas
  - Facade: BibliotecaDeReservas
  - SubClass: Reserva

- WebReservas / BibliotecaDePublicaciones / Publicacion
  - Client: WebReservas
  - Facade: BibliotecaDePublicaciones
  - SubClass: Publicacion

- WebReservas / BibliotecaDeCalificaciones / Calificacion
  - Client: WebReservas
  - Facade: BibliotecaDeCalificaciones
  - SubClass: Calificacion

STATE

- Reserva / EstadoReserva / Estados
  - Context: Reserva
  - State: EstadoReserva
  - ConcreteState: Pendiente – Concluida – Aceptada – Cancelada - Rechazada