Documentación TP Final del grupo JAVASTA de la materia POO II (UNQ)

Integrantes: Nicolás Garilli (garillinicolas@hotmail.com)
Camila Ruiz (camiruiz.cr21@gmail.com)
Iñaki Diaz (inaki.j.diaz@gmail.com)
Repositorio: (https://github.com/camiruiz/unq-po2-TPFinal.git)

Corrige: Fabrizio Britez

INTRODUCCION:

Teniendo en cuenta el apartado de introducción, creamos una clase usuario, en la cual utilizamos de template para las subclases propietario e inquilino. Además de esto, creamos la clase APP que representa nuestro sitio, todavía sin estar del todo seguros si la vamos a utilizar.

PUBLICACIÓN Y ALQUILER:

Leyendo este apartado logramos darle un poco de cuerpo a los usuarios, tal como su nombre, dirección de email y teléfono. Y también, creamos una clase inmueble con todas las características que estos deben tener. También tuvimos algunas dudas con respecto a los tipos como por ejemplo si los servicios iban a ser un diccionario o un array list.

Al día siguiente, nos dimos cuenta que era necesario que nuestro sitio web tenga publicaciones, por lo cual creamos su respectiva clase y lo llenamos con los datos que nos parecieron más relevantes, dejando así un poco más ligera a nuestra clase inmueble. Para hacer esto, decidimos dejar todas las características de nuestra entidad física de una propiedad en inmueble, y todas las características más relevantes a una publicación tales como el horario de checkin/checkout, precio por día, fotos, etc; dentro de nuestra clase publicación. Además de esto, hicimos que nuestra clase APP guarde dentro a todos nuestros usuarios como publicaciones.

BUSQUEDAS DE INMUEBLES

Filtros… con este enunciado tuvimos varias dudas a la hora de implementarlo ya que se nos ocurrieron muchas formas de resolverlo y no sabíamos cual era la más adecuada.

En un principio, pensamos en tener una clase SistemaDeBusqueda que trabajara con dos litsas, una con todos los filtros obligatorios que el usuario debía utilizar a la hora de realizar una búsqueda, y otra lista con los filtros “Opcionales” que el usuario podia utilizar si queria, pero haciendo esto nos dimos cuenta que no había forma de garantizar que todos nuestros filtros obligatorios estuvieran siendo utilizados por el usuario. También se nos ocurrió

que el SistemaDeBusqueda tenga un constructor para cada cantidad de filtros de búsqueda, pero de esta manera tendríamos un montón de constructores.
Por ultimo, nos decidimos de hacer la clase Busqueda, que representa una unica busqueda dentro de nuestro sitio, dicha busqueda tendra una lista de filtros y podra buscar por cada filtro en esa lista. Tambien tendremos la clase filtros la cual sera abstracta y les dara el comportamiento y estructura a cada filtro de nuestro sitio. Gracias a la ayuda del profesor Matias Butti, logramos cerrar la idea de los filtros obligatorios haciendo que nuestra clase filtros conozca a los TiposDeFIltrosEnum, que son ENUMERATIVOS que nos indican el tipo de cada filtro. Ademas de esto, agregamos un GestorDeFIltrosObligatorios, el cual conoce a los TiposDeFiltrosEnum y dentro guarda una lista con los filtrosObligatorios<TiposDeFiltros>. De esta manera, podemos preguntarle al
GestorDeFiltrosObligatorios cuales son los tipos de filtros que son obligatorios y
asegurarnos que nuestra busqueda contenga dichos filtros.

RANKING DE INMUEBLES Y PROPIETARIOS

Para solucionar este enunciado, se nos ocurrio realizar que tanto mis usuarios(El inquilino y el propietario) como mis publicaciones dispongan del colaborador interno CalculadorDeCalificaciones, el cual va a ser el encargado de calcularle
sus promedios de calificaciones, y va a guardar una lista con todas las calificaciones.
Una calificacion a su vez consta de una categoria y un puntaje.
Tambien le agregamos al CalculadorDeCalificaciones la responsabilidad de guardar una lista con todos los comentarios que va a poder brindarselos a los usuarios/publicaciones cuando se los pidan.

Un comentario consta de un cuerpo y un usuario que realizo el comentario.
Ademas de esto, agregamos la interfaz iPuntuable para darles protocolo a todas mis entidades que son puntuables (Usuarios y publicaciones).
Nota: Hicimos que la publicacion fuera la que se puede puntuar y no el inmueble, ya que consideramos al inmueble como la entidad fisica y la publicacion como la representacion de ese inmueble en la pagina, que es donde nos interesaria ver su puntaje.

VISUALIZACION Y RESERVA

Este enunciado nos parecio muy enfocado a todo lo que es el 'front end' por lo
cual no sabiamos si el Usuario deveria de tener el mensaje "Ver publicacion" porque ese mensaje solo tendria getters. Sin embargo, utilizamos este enunciado para agregar algunos mensajes, tales como que ahora la publicacion
debe de tener la informacion del dueNo (getInformacionDuenio).
Queda pendiente ver si tenemos que implementar los mensajes del usuarioPropietario tales como cuantas veces alquilo, cuantas veces fue inquilino y hace cuanto tiempo es usuario de la pagina. EL usuario debe tener estos getters o debe de existir algun objeto encargado de guardar esto?

CONCRECION DE UNA RESERVA
Nota: Ya para esta altura nos dimos cuenta que no era necesario tener una superClase abstracta Usuario, la cual tenga las subClases propietario e Inquilino, ya que como un usuario Propietario a la vez puede ser inquilino, es mucho mas comprensible que mi usuario "Base" sea el inquilino, y el usuario

propietario sea subClase de esta y agregue las funcionalidades extras que tiene el ser un usuario “propietario”.

Ahora si, con este enunciando nos dimos cuenta de la necesidad de, no solo tener reservas, sino tambien tener Solicitudes de reservas.
Primero, el usuarioInquilino puede solicitarReserva(), lo cual instancia una SolicitudDeReserva nueva, la cual nuestro usuarioPropietario, quien posee una lista con todas sus solicitudes, podra aceptar o rechazar.
Si el propietario decide aceptar una solicitud, dicha solicitud instanciara una Reserva, consolidando en nuestro sistema.

ADMINISTRACION DE RESERVAS PARA INQUILINOS

En base a este enunciado decidimos que nuestro objeto GestorDePublicaciones sea el encargado de implementar todos los metodos que este enunciado nos pide. En una primera instancia pensamos que el usuario sea el que implemente dichos metodos, pero debido a que el no es el encargado de realizar todos los calculos que estos metodos nos piden que hagamos, nos parecio mas apropiado que estos mensajes esten directamente en el objeto encargado de realizar todas estos filtrados, el GestorDePublicaciones.
ADMINISTRACION DEL SITIO

Para resolver este enunciado, primero se nos ocurrio crear un objeto nuevo que sea colaborador interno de cada entidad que requiera poder agregar servicios, tipos de categoria, etc.
Pasando en limpio, se nos habia ocurrido que por ejemplo, el inmueble conozca a su GestorDeTipoDeInmueble, el cual conozco la lista de

tiposDeInmuebles disponibles y puede agregar nuevos, los mismo para por ejemplo las publicaciones, que conozcan a su gestorDeServicios y este este encargado de manejar la lista de servicios disponibles y sea capaz de dar de alta nuevos servicios como pide el enunciado.
El problema con ese enfoque, el cual nos ayudo Fabrizio Britez que estaba mal, es que de esta manera estamos creando un monton de clases que no van a tener comportamiento alguno, vamos a tener objetos que lo unico que van a hacer es guardar una lista que otro objeto va a implementar. De esta manera estariamos “Sobre-diseñando” algo que no es necesario.
Para solucionar esto decidimos que todas estas listas que nos pide el enunciado, las guarde mi objeto APP (El cual representa nuestro sitio web).
No nos termina de convencer del todo esta solucion, pero sin duda pensamos que es mejor que la solucion que planteamos anteriormente.

POLITICAS DE CANCELACION

Para cumplir con este enunciado realizamos un template method, de esta manera, la politicaDeCancelacionDeReserva les proporciona comportamiento y protocolo a sus subclases de cancelacion. Ademas de esto, para resolver lo que plantea la CancelacionIntermedia decidimos implementar un patron state, el cual le dice a la CancelacionIntermedia cual va a ser su estado, si gratuito, MitadDePago o PagoTotal.

NOTIFICACIONES

Este enunciado no llegamos a plasmar nuestra idea en el codigo, pero si lo planteamos en el UML.
En base a este enunciado llegamos a la conclusion que ibamos a necesitar de un Observer que registre cuando ocurria un cambio en las publicaciones de nuestro sitio. Dichos cambios los denominamos “Eventos”. Planteamos la

existencia de tres eventos distintos, El evento de una nueva reserva, cuando se cancela una nueva reserva, y cuando ocurre una baja de precio de algun

inmueble. Y finalmente, estos eventos serian los encargados de notificar a sus interesados cuando ocurra un evento.

RESERVA CONDICIONAL

Este enunciado lo leimos al final de todo, y lamentamos no haber hecho una leida general de todo el trabajo antes de llegar a este punto, debido a lo que plantea este enunciado rompe un poco lo que veniamos implementando con las Solicitudes de reserva.
Para lograr solucionar este enunciado, deberiamos de tener una "ListaDeSolicitudesDeReservaCondicionales", la cual seria una queue de solicitudesDeReservas sobre la cual el sistema deberia de ser capaz de aceptar la primera en la cola en caso de que se cancele la reserva actual, y la primera en la cola pueda satisfacer la condicion de que los dias de reserva de la SolicitudDeReserva que esta en cola pueda entrar si hay suficiente dias de espacio disponibles habiendo cancelado la reserva actual.
Esta parte falta implementar, pero ya esta implementada una parte muy importante que es que las SolicitudesDeReserva deben de saber en que "estado" se encuentran, si las mismas estan aceptadas, pendientes o canceladas. Esto lo resolvimos con un state.

CORRECIONES DE TP:

Generales:

Ya arreglamos todo el código, los organizamos en subpaquetes y borramos todos los comentarios de codigo que habíamos dejado.

Precios y Calendarios:

TemporadaAlta y Reservas implementan la interfaz que interactúa con el metodo estaOcupadaEnFechas(LocalDate, LocalDate, IPeriodoDeTiempo).
de la clase Calendario.

Búsqueda

Tema filtros arreglado.

Ranking

Se arregló lo que rompía con el principio Open/Close en agregarPuntaje y agregarComentario en CalculadorDeCalificaciones.
Se arregló nuestro Usuario, que ahora tiene Propietario e Inquilino como subclases.

Concreción de Reservas

Arreglado el tema del mensaje que daba stack overflow delegando al estado la responsabilidad.
Ahora existe interacción con Solicitud y sus estados con los métodos aceptar y rechazar.

Política de cancelación

Arreglamos la interacción entre clases de Publicacion con Política de cancelación, y las Políticas con sus estados via el mensaje cancelarReserva(Reserva)

Gestión de publicaciones

No es singleton (ya hablado con Fabrizio) y ya eliminamos todo el codigo comentado.

Notificador

Implementado el observer. Nuestra publicación conoce a todos los que implementan la interface IListener. Los eventos son IListeners y cada respectivo evento conoce a quienes estan subscriptos a ese evento(implementando una interface que conoce cada Evento).

Reserva Condicional.

Implementado. Delegada la responsabilidad al EstadoDeSolicitudPendiente.

UML:

Arreglado lo de Usuario/Inquilino/Propietario.
Definidas las relaciones.

Tests:

Hechos.
Cobertura: 99.7%