Informe Trabajo Práctico Final

Integrantes:

1. Leandro Correa (drleandrocorrea@gmail.com)
2. Franco Paredes (francoismaelparedes@hotmail.com)
3. Alejo Perin (alejoperin11@gmail.com)

Decisiones de diseño

Se separó en paquetes el proyecto para una mejor comprensión del mismo. Se encuentra dividido en:

- domain
- implementations
- interfaces
- search
- views

Diseño UML

El diseño de clases UML fue realizado con plantUML, y exportado en formato SVG, por lo que podrá encontrar el mismo en el siguiente LINK:

- LINK DEL UML

Detalles de implementación y patrones de diseño

Publicación y Alquiler:

Se identificó una relación “es un” entre las entidades Propietario e Inquilino y la entidad Usuario. De ese modo, los primeros extenderán a este último, diseñado como clase abstracta.

Se identificó también la necesidad de tener una clase que represente un período de tiempo, la cual fue llamada DateRange.

Las demás clases siguen los requerimientos.

No se implementan patrones de diseño en este punto.

Búsquedas de inmuebles:

En el sistema de búsqueda de inmuebles se decidió utilizar una combinación de los patrones de diseño composite y builder. El Composite se utiliza para estructurar los filtros(ciudad, fechas, huéspedes, precios), donde:

- Cada filtro es una **leaf** diferente del composite.
- HousingSearch es el "**composite**" que aplica todos los filtros necesarios en secuencia.

A su vez, el patrón Builder permite construir la búsqueda agregando solo los filtros

relevantes definidos por el usuario. Así, HousingSearchBuilder añade los filtros opcionales según los parámetros proporcionados, construyendo una búsqueda personalizada que el sistema ejecuta para devolver los inmuebles que cumplen con los criterios establecidos.

Ranking de inmuebles y propietarios:

Se agregaron los tipos Rankeable y Ranker, el Ranker puede rankear a un objeto Rankeable y el Rankeable puede ser rankeado. La clase Ranking contiene la información para el ranking de categorías. No se utilizó ningún patrón.

Visualización y Reserva:

Se crearon clases que responderán distintos mensajes relacionados con la visualización de ciertos datos correspondientes a los usuarios a partir de las búsquedas con filtros. Su responsabilidad será calcular promedios, y adaptar la lógica de negocio a la de visualización.

Si bien no existía de acuerdo a los requerimientos la necesidad de adaptar una interfaz en específico, podría interpretarse como la implementación de un Adapter, donde el objetivo podría ser el protocolo de la clase HousingView, siendo a su vez su implementación el adaptador, mientras que el inmueble es el objeto adaptado.

Concreción de una reserva:

La reserva estará a cargo de la clase Booking. Para la implementación de este punto se sigue el patrón de diseño observer, siendo los participantes y roles:
Observable:

Clase Booking, implementa interfaz de Observable

Observers:

Clases ConfirmationEmailSender y BookingSystem, quienes implementan BookingAcceptedObserver.

Administración de reservas para inquilinos:

La administración de reservas está a cargo de BookingSystem quien tiene sus observadores dependiendo si se cancela o acepta una reserva por parte del inquilino. Para esta solución se utilizó el patrón Observer para notificar a los observers y enviar los emails cuando sucede la cancelación o confirmación a través del EmailSender

Observable: BookingSystem

Observers: ConfirmationEmailSender, CancellationEmailSender estas clases implementan BookingAcceptedObserver y BookingCancelledObserver

Administración del Sitio:

El sitio lo administra la clase Admin donde podrá dar de alta un nuevo tipo de propiedad, servicios, categorías, listados de gestión.

No se implementan patrones de diseño para este requerimiento.

Políticas de cancelación:

En este sistema, se implementa el patrón Strategy para manejar las distintas políticas de cancelación, permitiendo que cada propietario elija la política que se aplica a su propiedad. Cada política de cancelación: "gratuita hasta 10 días", "sin cancelación", o "intermedia", se representa como una estrategia diferente, permitiendo que el sistema gestione el cobro de manera flexible según la política seleccionada. Esto facilita la incorporación de nuevas políticas en el futuro sin modificar la estructura principal del sistema, ya que solo sería necesario agregar una nueva estrategia.

La cancelación de la reserva se debe realizar partiendo del booking system, donde al cancelar la reserva se van delegando las acciones necesarias hasta llegar al propietario y al huésped que hizo la reserva, afectandolos a ambos dependiendo de la estrategia elegida a la hora de crear la propiedad.

Roles:

- **Context**: Housing
- **Strategy**: CancellationPolicy
- **ConcreteStrategy**: IntermediateCancellation, FreeCancellation, NoCancellation

Notificaciones:

Se creó un publicador/manejador de eventos, donde los observadores se suscriben a los eventos que quieren.

Para esta implementación se usó el patrón Observer

Roles:

- **Subject**: EventPublisher
- **Observer**: Web externa, App Mobile externa

Reserva condicional:

Se agregaron dos estrategias en BookingSystem para que se use una u otra dependiendo si la propiedad ya está alquilada. En caso de estar alquilada se agrega la Booking a la cola de reservas condicionales para esa propiedad, y si está libre se agrega a la lista de reservas confirmadas.

El patrón que se utilizó fue el Strategy

Roles:

- **Context**: BookingSystem

- **Strategy**: BookingStrategy
- **ConcreteStrategyA**: BookingConfirmedStrategy
- **ConcreteStrategyB**: BookingConditionalStrategy