Grupo 19

Integrantes:

Bastián Bernal Villegas

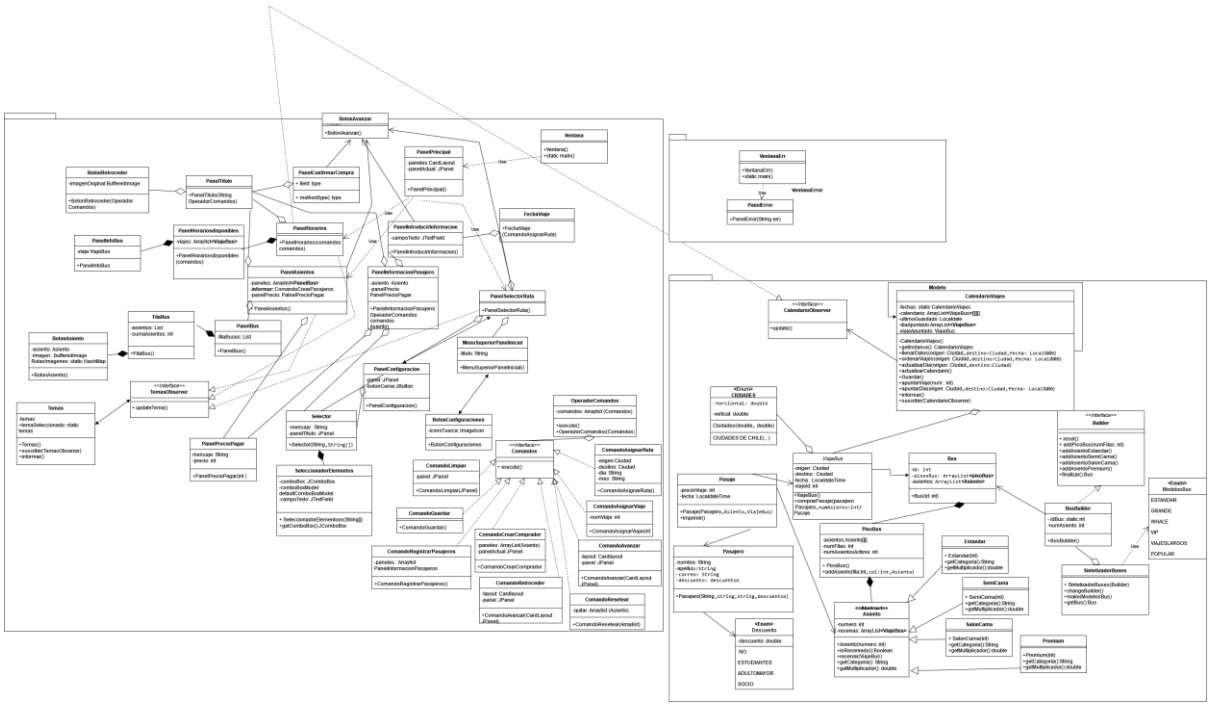
Andrés Chaparro Maldonado

Esteban Navarrete Mella

Enunciado **:Sistema de reserva de asientos de autobús**

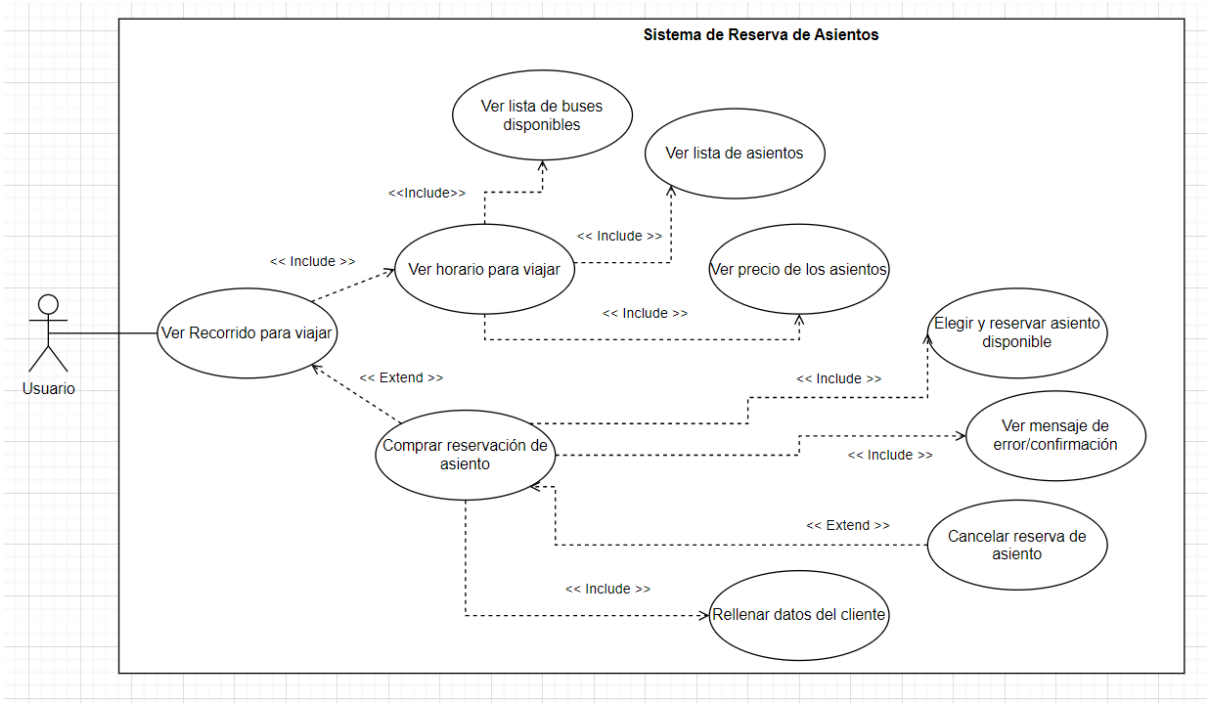
- El sistema de reserva de asientos de autobús permite al personal de una empresa de autobús elegir y reservar asientos de forma conveniente por su cliente. Los usuarios pueden visualizar una representación gráfica de los asientos disponibles en el autobús y seleccionar los que deseen ocupar. El sistema muestra información detallada sobre cada asiento, como su ubicación, número y categoría (por ejemplo, semi cama, Salón Cama).
- Una vez que los usuarios seleccionan los asientos deseados, el sistema verifica la disponibilidad y permite confirmar la reserva mostrando el precio a pagar. En caso de que algún asiento ya esté reservado por otro pasajero, se informa al usuario para que pueda elegir otro asiento disponible. El personal confirma el pago (no gestionado por el sistema) lo que reserva los asientos.
- El sistema debe gestionar varios tipos de autobuses (por ejemplo, con diferente número de plazas, o de 1 o 2 pisos...).
- El sistema debe mostrar un menú que permita seleccionar el autobús en función de su horario y recorrido (se supone que estos datos están disponibles con los autobuses vacíos cuando se lanza el software)

Para este problema planteado creamos el siguiente el diagrama UML para postular su solución (Se puede visualizar mejor desde el repositorio):



Utilizamos 3 packages: Uno para la lógica, otro para las vistas gráficas y otro para crear ventanas por excepciones (errores)

Para el diagrama de casos de uso junto con el referente llegamos a la conclusión de que el diagrama adecuado estaría dado por el siguiente diagrama:



Patrones

Para el caso de los patrones fue lo primero en lo que pensamos antes de empezar a programar, priorizamos esto ya que nuestra idea era programar orientado a los buenos patrones de diseño y no era ideal que a mitad del código forzáramos un cambio. Los patrones que escogimos fueron:

1)**Builder:** En nuestro problema estaba presente implícitamente la existencia de un objeto Bus, sin embargo, al pensar en cómo sería la clase Bus y su constructor sobre todo era un poco atemorizante pensar en cómo íbamos a hacer que existieran diferentes modelos de bus. Por suerte nos encontramos con este gran patrón que nos dio la solución a este problema, con su clase Builder la construcción de un bus era totalmente personalizable y además con su clase directora teníamos la forma perfecta de tener nuestros propios modelos de bus.

2)**Singleton:** Bueno, en nuestro problema se nos habla de una app, para guardar todos los viajes posibles es evidente que íbamos a necesitar una "base de datos" y singleton de hecho en la misma página menciona que sirve para bases de datos. Simplemente con este patrón nos asegurábamos de que existiera una única base de datos y nos facilitó el acceso a esta.

3)**Command:** En la tarea 3 llego a ser algo tedioso tener que pasar tantos objetos por parámetros para llegar a un listener que le diera un uso al fin. Nosotros encontramos el patrón command que nos sirvió para hacer una capa entre las instrucciones y la gráfica, gracias a esto pudimos separar mejor el trabajo y luego unirlo de una forma más limpia las vistas con la lógica.

4)**Observer:** Este patrón no teníamos planeado usarlo en un principio, pero en un momento de necesidad nos llegó como solución a uno de los problemas que teníamos que enfrentar, hubo un punto en el que se haría algo complejo actualizar los paneles según ciertas propiedades del Calendario (Base de datos), por lo que utilizamos un observer hacia este de forma que ahora los paneles solo cambiarían cuando escuchen el cambio dentro de nuestros datos. Adicionalmente nos sirvió para poder cambiar el tema de la interfaz gráfica en todos los paneles

Resultados de la interfaz:

Primero que nada, el programa se corre desde la clase Ventana entregando este primer resultado (El tema se escoge al azar):

Al escoger una ruta y fecha correctas podemos avanzar con ver horarios

Seleccionamos un viaje disponible:

Escogemos N asientos y se generaran N paneles:

Donde llenamos la información.

Finalmente se confirma la compra y se “imprimen” los pasajes (se genera un archivo txt en la carpeta correspondiente.

Decisiones tomadas

Las primeras decisiones estuvieron orientadas a la elección de patrones, por ejemplo, en un inicio queríamos usar un holder parar el cambio de paneles, pero más tarde descubriste que ya existía la layout CardLayout asi que en base a eso tuvimos que decidir como emplearíamos nuestro código para cumplir con los requisitos.
Luego tuvimos que tomar las decisiones sobre las mecánicas que tendría la app, por ejemplo, en base a que se crearían los viajes, o si debiese haber diferentes marcas de bus, etc., todas estas dudas se fueron consultando con el referente según lo que a él le pareciera más conveniente para sus necesidades.
También algo en lo que teniamos bastante duda y al final tomamos la decisión solos era el hecho de si debería existir la clase bus o no, ya que como tal los buses no los usamos, sino que al menos para esta app nos importa directamente el piso del bus para graficarlo, entonces teníamos la pregunta si una clase bus era necesaria, o si la clase “ViajeBus” ya contaba como el bus en sí. Al final nuestra decisión fue crear la clase y separar el bus del viaje pensando siempre en la abstracción y también en el principio de responsabilidad única, también por eso terminamos usando el patrón Builder, siempre pensando en las posibles actualizaciones que podría tener la app algún día y en facilitar el hecho de querer añadir cosas nuevas en un futuro.

Problemas encontrados

Los primeros problemas fueron principalmente la falta de entendimiento entre algunas ideas, como fue en el caso del UML base que fue hecho para servirnos de guía y cumplir con los patrones, sin embargo, pasaba que a veces por la falta de especificaciones en esta primera versión del UML los interpretábamos de diferentes maneras ralentizando la programación en la lógica.

Luego se nos complicó la visualización principalmente del piso de bus ya que tuvimos la maña de querer que los asientos de mayor categoría ocupen más espacio. Al final logramos solucionarlo de una forma algo compleja y poco optima a nuestro parecer (Notara que al abrir el panel de asientos tardara más en cargar) Intentamos solucionar

este problema usando GridBagLayout pero por falta de tiempo y experiencia usando esta layout no logramos manejarla correctamente por lo que se descartó usarla por esta vez.
También notamos que había problemas con una de las imágenes de los asientos, esto entre otros bugs que no alcanzamos a corregir en el plazo, pero aun así dejamos arreglado fuera de plazo para que se pueda ocupar la app más cómodamente
Al final consideramos que nos faltó centrarnos más en objetivos concretos de a poco ya que al final logramos hacer una app con funciones extra pero no logramos solucionar el retraso de carga entre otras cosas a tiempo.

Para más información de ciertas mecánicas y funcionalidades de la app las dejaremos en el README