Trabajo Práctico Especial
Primera Parte

Profesores: Gellon, Ivonne.
Guccine, Leonel.
Lazurri, Guillermo.

Grupo: 9.

Fecha de entrega: 05/05/2024.

Integrantes: Bianchini, Pilar.
Corvalan, Agustin.
Santiago, Felipe.
Stadler, Laura Maria.

Introducción:

Para este trabajo práctico, debíamos construir un sistema informático para una empresa de traslados de pasajeros. La interfaz de usuario será una computadora, los requerimientos serán limitados. En dicho sistema, seremos capaces de hacer altas y modificaciones en nuestros vehículos y choferes de los cuales la empresa dispone. También podremos administrar los pedidos que llegan a la empresa y generar viajes a partir de ellos. En función de todos estos elementos, podremos generar diferentes informes y reportes como listado de clientes, sueldos, etc.

Arquitectura en Capas:

Implementamos este sistema a partir de un desarrollo estructurado en capas, consistiendo en las capas de Presentación (en esta primera parte será simplemente la clase main, luego se implementaran ventanas) y capa de Negocios (la cual consiste el resto de nuestras clases, en la cual interactúan entre sí para brindar lo solicitado devuelta a la capa de presentación)

Funcionamiento del Sistema:

Desde el main, primero se genera mi “flota” de tanto vehículos como choferes, los cuales serán usados para realizar distintos viajes. Después, se empiezan a generar pedidos de distintos clientes, y se pide al Sistema que genere un viaje a partir de ellos. Dicho pedido puede ser rechazado por distintas razones, puede que sea invalido, que la empresa no tenga choferes disponibles, vehículos disponibles o ningún vehículo cumpla los requisitos del pedido.

Cada vehículo tendrá una prioridad calculada, en función de los requisitos del pedido (algunos vehículos serán más aptos para algunos pedidos que otros). Luego, retiró al vehículo con mayor prioridad y al chofer asignado, de un estado de disponible a no disponible. A partir de esto mi viaje “comienza”, se calcula su costo y al terminar se vuelven a marcar tanto al vehículo como al chofer en disponibles.

El sistema también tiene diferentes funcionalidades como altas de choferes y vehículos, reportes de sueldos de un chofer, viajes de un cliente entre determinadas fechas, etc.

Variables estáticas:

Las variables que consideramos que tenían que ser estáticas, fueron en las distintas clases de tipos de chofer, dado que los pluses iban a ser los mismos para todos los choferes, ya que resultaba más sencillo el cálculo del sueldo si estas eran variables estáticas de clase.

También en la clase abstracta viaje, consideramos que el precio base debía ser una variable estática de la clase, porque este precio es el mismo para todos los tipos de vehículo.

Patrones de diseño aplicados:

Patrón Factory: Este patrón creacional fue aplicado a la hora de solucionar nuestro problema de creación de objetos del mismo tipo pero distintas clases finales. Implementamos un FactoryVehiculos, para desacoplar al sistema de la responsabilidad de crear nuestros distintos vehículos de clase final moto, auto, o combi. Lo implementamos de igual manera con un FactoryViajes, el cual se responsabiliza de crear un IViaje, con su respectivo costo calculado.

Patrón Decorator: El patrón Decorator fue implementado bajo el ámbito de FactoryViajes, de manera que podamos solucionar nuestra explosion de clases generada por las diferentes combinaciones resultantes entre cada viaje y sus diferentes acepciones en términos de baúl, mascotas y zonas.

Patrón Singleton: Singleton fue aplicado en diversos sectores del código para asegurarnos que ciertos objetos sean instanciados solo una vez, por ejemplo, las distintas Factory o el mismo objeto Sistema.

Patrón Facade o “Fachada”: Para que la capa de presentación se pueda comunicar con la capa de negocios, utilizamos un patrón Facade para instanciar un objeto de tipo Sistema que actuará como “telón” o fachada para el usuario, haciendo que todas las solicitudes pasen por ella.

Patrón Template: Dicho patrón fue usado a la hora de calcular la prioridad de cada vehículo antes de ser seleccionado. Como los pasos a seguir para el cálculo se repetían, usamos Template para hacer que cada vehículo implementara cada paso como debía, y dichos métodos fueron invocados paso a paso, independientemente de que tipo de vehículo era finalmente.

Consideraciones:

Clase viaje: decidimos no poner el atributo fecha, siendo algo que lo obtiene del atributo pedido.

Fechas: el uso de fechas lo hicimos a partir de la clase GregorianCalendar.

Uso de excepciones: utilizamos 5 excepciones para manejar los posibles errores en el programa:

- ZonaInvalidaException: es lanzada por el Factory de Viajes cuando la zona es incorrecta.
- VehiculosNoDisponiblesException: es lanzada cuando no se encuentra un vehículo que cumpla con el pedido.
- ChoferNoDisponibleException: es lanzada cuando no hay ningún chofer disponible en la lista.
- PedidoNoValidoException: es lanzada cuando el pedido no es correcto.
- ViajeNoPosibleException: es lanzada cuando el viaje no se puede realizar, siendo esta excepción el padre de las anteriores.

Uso de interfaces: utilizamos una única interfaz IViaje, para que los Decorator tengan el mismo comportamiento que los viajes.Independientemente de si mi factory genera un Viaje decorado o no, mi manejo de objetos serán de tipo IViaje.

Decorator: La consigna indica que si un viaje no necesita baúl o mascota, estos estados no agregan ningún tipo de modificación al cálculo del costo, lo cual haría que nuestros decoradores “SinMascota” y “SinBaul” no tengan mucho sentido, ya que devuelven al mismo encapsulado tal como estaba. Sin embargo, dejamos la estructura de ambos decoradores ante la posible situación que se aplique algún descuento o modificación del costo en estos escenarios en el futuro de la empresa.

Clonación: Debimos hacer diversos casteos a IViaje dentro de la clonación profunda.

Observaciones finales:

Como grupo, nuestra primera dificultad fue cómo íbamos a encarar y unir todos los temas vistos a lo largo de esta primera parte.

Luego cuando empezamos con la resolución del problema, encontramos complicado como realizar la creación del viaje a partir del pedido solicitado, como plantear el uso del patrón Decorator con el uso de la interfaz IViaje.

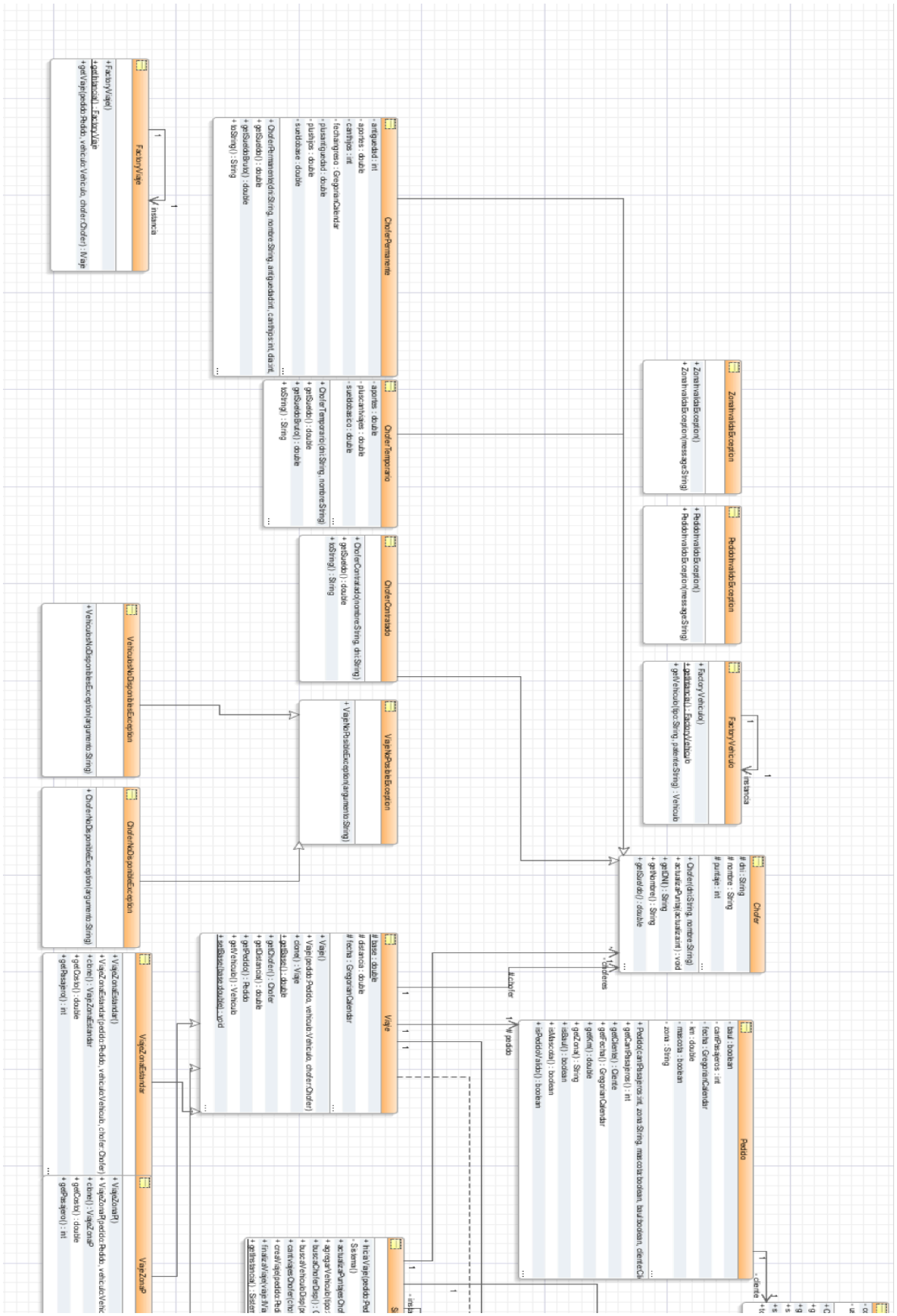
También nos resultó una complicación todo lo relacionado a excepciones, quien debía lanzarlas, donde las atrapamos y que tipo de excepciones diferentes debíamos lanzar.

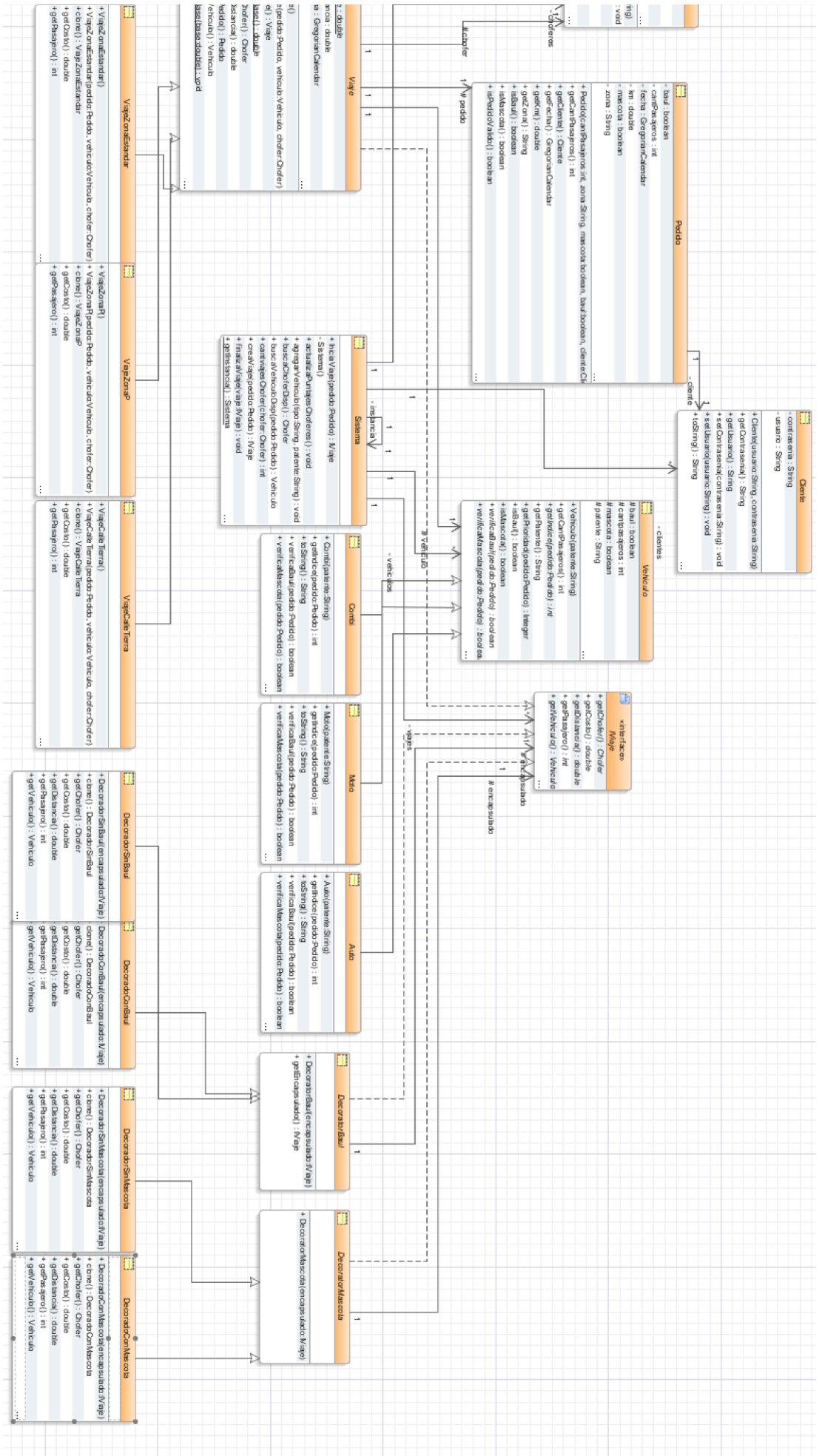
Manejarnos con pull / push requests con nuestro repositorio GitHub al principio generó un poco de complicaciones, pero luego, rápidamente nos dimos cuenta de su gran utilidad para codear en equipo mediante el uso de un repositorio.

Dentro de todo, como grupo estamos orgullosos de cómo nos quedó nuestro FactoryViajes, en donde pudimos implementar el patrón Decorator que nos costó mucho entender su uso. Logramos combinar los patrones Factory y Decorator y generar un Factory que devuelva viajes ya decorados internamente.

Lo que más nos gustó de esta primera parte del trabajo fue el hecho de poder implementar todos los patrones vistos en teoría en ejemplos concretos que realmente nos solucionaban problemas en la implementación de nuestro sistema informático.

Diagrama de clases / UML :




Cliente
Pedido
Vehículo
Sistema
Viaje
«interface» IViaje
Combi
Moto
Auto
ViajeZonaEstandar
ViajeZonaP
ViajeCaleTierra
DecoradorSinBaul
DecoradorConBaul
DecoradorSinMascota
DecoradorConMascota
DecoradorBaul
DecoradorMascota