INFORME TRABAJO PRÁCTICO:
“SUBÍ QUE TE LLEVO”

Asignatura: Programación 3

Integrantes:

- **Gonzalez Facundo Nehuen**
- **Santin Fransisco**

Informe:

Acerca del proyecto:
Para realizar este proyecto de un servicio de gestión de Clientes, Choferes, Vehículos y Viajes, tuvimos que pasar por distintas etapas.
La primera etapa fue la realización del diagrama UML, con el cual pudimos definir las dependencias de las distintas Clases y tener un “pantallazo” general de todo el proyecto, también era más simple la división de tareas una vez hecho todo el esqueleto del proyecto.
Lo segundo fue empezar a implementar los distintos Patrones de diseño aprendidos en clase, y un simple testing del esqueleto del sistema para no arrastrar errores de compilación a futuro.
Mientras avanzabamos en el proyecto fuimos modificando la estructura del proyecto, para implementar el patrón Factory y el Patrón Decorator correctamente. Más adelante utilizamos estos patrones que son necesarios para realizar la solicitud del pedido del cliente correctamente, este “Pedido” se vuelve un atributo del Viaje, se le asigna un vehículo apropiado (Prioridad del vehículo Patrón Template) y se le asigna un Chofer Disponible.
Desde este punto el viaje se considera iniciado, y al finalizar se realiza el pago del mismo. Una vez pagado, el cliente califica al chofer y el viaje va a una lista de Viajes realizados en el sistema, esta lista se utilizará para el cálculo de los salarios de los choferes contratados y choferes temporarios, los choferes permanente reciben una cantidad fija de sueldo todos los meses

FUNCIONALIDADES DEL SISTEMA:
Administrador:
-Altas, modificaciones y consultas de clientes, choferes y vehículos.
-Listados de clientes, choferes, vehículos y viajes.
-Salarios mensuales de cada chofer.
-Cálculo del total de dinero necesario para pagar los salarios.
Clientes:
-Registro de nuevo cliente (no repetir nombre de usuario).

-Formulario de solicitud de viaje: Informar si se acepta o los motivos del rechazo.
-Pagar viaje.
-Calificar chofer.
-Historial de viajes.

PATRÓN FACTORY Y DECORATOR:
El patrón Factory es utilizado en la clase FactoryVehiculos y se basa en la creación de un Vehículo apropiado que depende del Tipo(“Moto”,”Automovil”,”Combi”) y de la patente, este método solo crea los vehículos que se utilizaran en el sistema, es decir, no se utiliza para gestionar la disponibilidad de los mismo al realizar los viajes.
Tanto El Patrón Factory como El Patrón DECORATOR es utilizado en la clase Factory Viajes ya que dependiendo de la Zona Del Pedido(Para la Zona utilizamos la clase Enum, para mayor claridad en la utilización de las Zonas requeridas Zona Estándar, Zona Calle sin Asfaltar, Zona Peligrosa) se generará una clase concreta distinta, estas clases concretas generarán un costo adherido distinto al viaje. Después esta Zona es “Decorada” con las Clases “Viaje Con Baul “y “Viaje Con Mascota”. dependiendo si en el pedido el atributo de la mascota y el baúl son verdaderas o falsas.

PATRON TEMPLATE:
Se aplica el patrón template en la clase Vehiculo. El método disponible para el cliente (público) es getPrioridad. Este método se implementa en la clase abstracta Vehiculo y los métodos que se invocan dentro de este son verificarBaul, verificarCantPasajeros, verificarPF y calculoPrioridad. Estos cuatro métodos son implementados en las clases concretas y de ese modo se logra el polimorfismo.

PATRON SINGLETON:
Este patrón lo utilizamos en la Clase Sistema ya que no puede haber más de una Instancia de la misma, y facilita la accesibilidad de sus atributos en otras partes del sistema, además de facilitar un único punto de acceso a todo el sistema.

EXCEPCIONES:
Implementamos las siguientes excepciones que se utilizan al procesar el pedido y al crear un cliente:

- PedidoInvalidoException:se utiliza al procesar el pedido para verificar que todas las variables son válidas.
- SinChoferDisponibleException:esta excepción ocurre cuando hay un pedido y no hay un chofer disponible para ese pedido.
- SinVehiculoDisponibleException:esta excepción ocurre cuando hay un pedido y no hay un vehiculo disponible para ese pedido.
- UsuarioRepetidoException:esta excepción ocurre cuando se ingresa un nombre de usuario que ya existe en la lista de clientes al agregar un Clase de tipo Cliente.

TOMA DE DECISIONES:

Cliente y Administrador:

Para las Clases de Cliente y Administrador decidimos tomar los atributos comunes de estas dos, Nombre de usuario,Nombre Real y contraseña, crear una clase abstracta común llamada Usuario.
Después separamos las responsabilidades/métodos del cliente y administrador respectivamente.

Administrador:

este se encarga de:

Altas: agregar Choferes, Clientes y Vehículos a las respectivas listas del sistema,se utiliza el método de alta correspondiente con sus parámetros, y devuelve un objeto del tipo correspondiente.
En la alta de vehículos utilizamos el patrón Factory.
En la alta de Choferes tuvimos que separar los métodos(AltaChoferPermanente,AltaChoferContratado,AltaChoferTemporario), ya que depende de qué tipo de Chofer que se quiere, se agregan los atributos necesarios para el constructor.

Bajas: eliminación de Choferes, Clientes y Vehículos, se busca por un parámetro de entrada, DNI para Choferes,nombre de usuario para clientes y patente para los vehículos y se busca el objeto que coincide el atributo propio con el parametro de entrada y se elimina de la lista correspondiente.

Modificaciones:se utiliza igual que en la eliminación un atributo a buscar y además se otorgan atributos nuevos a modificar, es decir, por ejemplo: en las modificaciones de Cliente se requiere un nombre de usuario a buscar, y los datos nuevos a reemplazar, nuevo nombre de usuario,nuevo nombre y nueva contraseña.
Consulta:parecido a las bajas o modificaciones, se escribe un parámetro de entrada a buscar en la lista correspondiente y devuelve el objeto adecuado.

Listados: devuelve el arrayList de Choferes,Vehículos,Viajes y Clientes.

Salario Mensual:dando como parámetros un chofer y una Fecha(para la fecha solo se utiliza el mes y año en cuestión) se calcula el salario mensual del chofer en cuestión, para esto utilizamos un IF para obtener la Clase Concreta del Chofer y obtener el Sueldo Bruto de ese mes.

Dinero Necesario: dando como parámetro una Fecha de la cual se utilizara el mes, este método recorre la lista de Choferes y va sumando el sueldo bruto del Chofer, para obtener el total de dinero necesario que necesita la empresa para pagar a todos sus choferes ese mes.

Cliente:

este se encarga de:

Solicitar un Viaje: dado como parámetro un objeto de tipo Pedido.
Este llama a un método del Sistema que procesa el pedido dado.

PagaViaje: dando como parámetro un Viaje abstracto, este llama a un

Método del sistema que procesa el pago en cuestión.

Califica Chofer:se llama a un método del sistema que califica al chofer.

Estos métodos se explicaran en la parte del **Sistema**.

Vehículos

para los vehiculos tambien creamos una clase abstracta que tiene los atributos comunes de todos los tipos de vehículos utilizados, y derivan en sus clases concretas, cada clase concreta tiene un constructor con las especificaciones correspondientes, es decir si es moto la cantidad de pasajeros máxima es 1, no tiene baul y tampoco es apto para mascota, si es automóvil la cantidad de pasajeros máxima es 4, si tiene baul y es apto para mascota, y si es combi la cantidad de pasajeros máxima es 10 y no es apto para mascota pero si tiene baul.

Además se verifica para el patrón template si cada uno de sus atributos es correcto y si no lo es, devuelve un “false”.

Chofer

Al igual que para los vehículos separamos los atributos comunes de los choferes y creamos la clase Chofer abstracto, y también creamos la Clase Chofer Empleado que tiene los atributos comunes del chofer Temporario y el Chofer Permanente. la clase Chofer ademas tiene dos metodos abstractos llamados getSueldBruto y getSueldoNeto. que utilizamos para el chofer Permanente y Temporario.

Chofer Permanente: este tiene los atributos estáticos del porcentaje de antigüedad y el porcentaje de plus X Hijos, ya que son invariantes para todos los choferes que sean Permanentes. Además tiene el atributo Fecha de ingreso que se utilizara para calcular la antiguedad del chofer.

El sueldo Bruto del Chofer se calcula multiplicando el sueldo basico por el plus por antiguedad(es el plus por la diferencia de anios entre la fecha actual y la fecha de ingreso)

El Sueldo Neto se calcula como el SueldoBruto menos un porcentaje de aportes.

Chofer Temporario: Para el cálculo del sueldo del Chofer Temporario decidimos agregar una variable más llamada Cantidad de viajes, este atributo se modifica en el SetCantidadViajes con su parámetro de entrada “date” que sería el mes del que se quiere obtener la cantidad de viajes que realizó ese chofer, así podemos calcular el sueldo Bruto del mes en cuestión.

El sueldo Neto se calcula de la misma manera que el Chofer Permanente.

Chofer Contratado:A diferencia de los otros dos Choferes que tienen un sueldo bruto y un sueldo neto, el chofer contratado no posee ningún aporte ni un sueldo básico, su sueldo se calcula con un método get Sueldo Bruto que necesita como parámetro una fecha(LocalDate date) y se utilizará para recorrer la lista de viajes y ir sumando un porcentaje del costo de los viajes que realizo ese chofer en ese mes.

Viaje:

Para el viaje al igual que para los vehículos decidimos crear una clase abstracta llamada Viaje Abstracto que tiene como atributo el Pedido,El chofer y el vehículo, además del costo base los viajes y la distancia recorrida en el viaje, también esta clase implementa una Interfaz Iviaje que tiene los métodos: getCosto,getDistanciaRecorridaEnKm y getCantPasajeros, además extiende de cloneable para poder clonar la arraylist de viajes en el sistema.

Para las clases Concretas decidimos utilizar atributos estáticos para el cálculo del costo por pasajeros y por Kilometros, asi son fácilmente manipulables si estos varían en un futuro.

ViajeDecorator:

Al crear un viaje, siempre hay una clase correspondiente a una zona (ViajeEstandar, ViajeZonaPeligrosa, ViajeCalleSinAsfaltar). Pero para el caso de viajes con mascotas y con baúl se decidió usar el patrón Decorator. Las ventajas de usar este patrón son:

- Se puede extender el comportamiento de ViajeAbstract sin crear subclases.
- Se pueden añadir y remover responsabilidades en tiempo de ejecución.
- Se pueden combinar diferentes comportamientos “envolviendo” una instancia con múltiples ViajeDecorator.
- El Principio de Responsabilidad Única: Se divide una clase que implementa posibles variantes en varias clases más pequeñas.

Por ejemplo, si más adelante se define un comportamiento distinto para los viajes con bebés, todo lo que hay que hacer es crear una clase ViajeConBebes que extiende ViajeDecorator.

ViajeDecorator extiende ViajeAbstract. Al “envolver” una instancia de tipo ViajeAbstract con un ViajeDecorator, el cliente no verá ninguna diferencia.

Sistema:

El sistema se encarga de todas las funcionalidades requeridas para el proyecto además de utilizar el patrón Singleton para facilitar la accesibilidad de los datos a todo el proyecto.

Los atributos del Sistema son los ArrayList de Choferes,Vehículos,Viajes y Clientes. Sin Tener en cuenta los métodos de Getters, la agregación de elementos a las distintas Listas, a continuación se explicará más en detalle los métodos relevantes del **Sistema**:

procesarPedido: Cuando el cliente realiza una solicitud de viaje, este es el método del Sistema que se encarga de verificar que el pedido sea válido (que existan vehículos capaces de satisfacer el pedido) y de asignar un vehículo y un chofer al viaje.

Para conseguir el vehículo, primero se consigue una moto, un automóvil y una combi de los disponibles y luego se pasan como parámetro a la función vehiculoConMayorPrioridad.

Lanza las excepciones PedidoInvalidoException, SinVehiculosDisponiblesException y SinChoferDisponibleException, de modo que se puede generar un mensaje a partir de esas excepciones para mostrar los motivos del rechazo.

Una vez creado el viaje se agrega al listado de viajes (ArrayList).

vehiculoConMayorPrioridad: Se usa el método getPrioridad() de la clase Vehículo para obtener el vehículo de mayor prioridad de los pasados como parámetro (moto, automóvil y combi).

procesarPago: Se invoca cuando el cliente paga el viaje. Todavía no está implementada.

guardarCalificacionDelChofer: Se invoca cuando el cliente califica al chofer. Todavía no está implementada.

registrarViajeFinalizado: El chofer se ocupa de registrar el viaje como finalizado. En ese momento se invoca esta función para enviar el vehículo y el chofer al final de sus listas y marcarlos como disponibles.

listadoViajes: Se devuelve una lista ordenada de mayor a menor por costo. Para ordenar la lista se usa quicksort, y para no alterar la lista original se crea un clon. Para eso se implementa la interfaz Cloneable en ViajeAbstract y en Pedido (porque Pedido es la única relación de composición que tiene ViajeAbstract). También se implementa la interfaz Comparable en ViajeAbstract para comparar por costo.