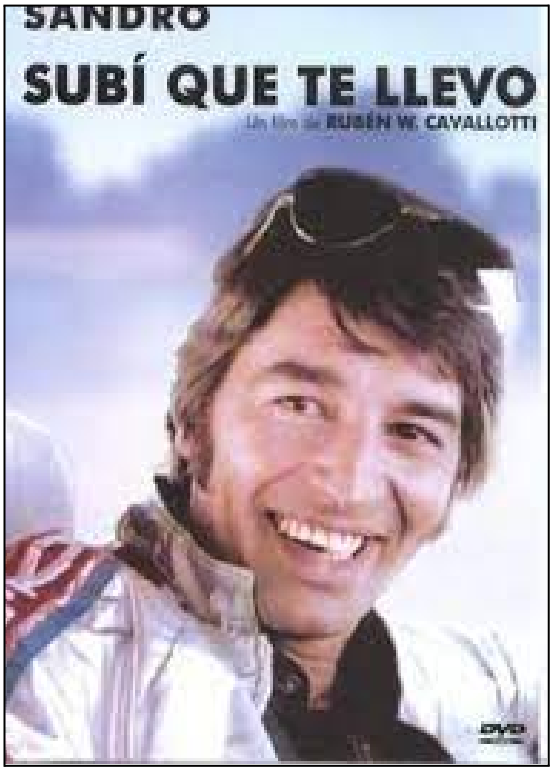
PROGRAMACIÓN III:

TRABAJO PRÁCTICO

GRUPO 1

PARTE 1



Mayor, Dario Agustín.

Nigro, Maite.

Patriarca, Martin.

Verón, Francisco.

Se explican brevemente las clases intervinientes en el programa.

- **Excepciones:**

ChoferNoDisponibleException*:* Excepcion que lanzamos cuando todos los choferes están ocupados y no se le puede asignar un pedido a un chofer porque no hay ninguno disponible

PedidoImposibleException*:* Excepcion padre que lanzamos cuando no es posible tomar un pedido por diferentes motivos como falta de vehículo,chofer o pedido incoherente

PedidoIncoherenteException*:* Excepción que lanzamos cuando el pedido solicitado por el cliente es incoherente en relación a los servicios brindados por la empresa

UsuarioRepetidoException*:* Excepción que lanzamos cuando un usuario nuevo se quiere registrar con un nombre de usuario que ya está siendo utilizado por otro usuario

VehiculoNoDIsponibleException*:* Excepcion que lanzamos cuando todos los vehiculos están siendo usados y no hay ninguno disponible para seguir tomando pedidos

- **Interfaces:**

IVehiculo*:* Interfaz que define los métodos que la clase Vehículo debe implementar

IViaje*:* Interfaz que define los métodos que la clase viaje debe implementar

- **Clases:**

Sistema*:* Clase que representa a la empresa en el sistema

Administrador: Clase a utilizar para cuando el usuario se registra como administrador.

Empleado*:* Es la clase que representa a todo empleado de la empresa.

Chofer*:* Clase abstracta que representa a los choferes de la empresa y se extiende de la clase empleado

ChoferContratado*:* Clase extendida de la clase Chofer y representa a los choferes de tipo Contratado

ChoferPermanente*:* Clase extendida de la clase Chofer y representa a los choferes de tipo Permanente

ChoferTemporario*:*Clase extendida de la clase Chofer y representa a los choferes de tipo Temporario

Usuario*:* Clase abstracta que permite usuarios nuevos en el sistema

Cliente*:* Clase que representa el cliente y permite crear un nuevo cliente en el sistema con toda su información.

Dirección*:* Clase que representa la dirección donde vive el cliente.

VehiculoFactory*:*Clase utilizada para verificar si el vehículo a utilizar será un auto,combi o moto

Vehiculo*:* La clase implementa la interfaz Vehículo y es la clase padre de todos los vehículos que tiene la empresa

***Tutu**:* Clase que se extiende de Vehículo y es utilizada para calcular la prioridad que tendrá el **auto** a la hora de escoger un vehículo

***Moto**:* Clase que se extiende de Vehículo y es utilizada para calcular la prioridad que tendrá la moto a la hora de escoger un vehículo

***Combi**:* Clase que se extiende de vehículo y es utilizada para calcular la prioridad que tendrá la combi a la hora de escoger un vehículo.

***DecoratorEquipaje**:* Clase que se usa para agregar funcionalidades adicionales al equipaje de manera dinámica.

***DecoratorMascota**:* Clase que se usa para agregar funcionalidades adicionales a la mascota de manera dinámica.

***DecoratorZonas**:* Clase padre que se usa para agregar funcionalidades adicionales a las zonas de manera dinámica.

***DecoratorZonaEstandar**:* Clase que se extiende de DecoratorZonas y se usa para agregar funcionalidades adicionales a la zona estándar de manera dinámica.

***DecoratorZonaPeligrosa**:* Clase que se extiende de DecoratorZonas y se usa para agregar funcionalidades adicionales a la zona peligrosa de manera dinámica.

***DecoratorZonaSinAsfaltar**:* Clase que se extiende de DecoratorZonas y se usa para agregar funcionalidades adicionales a la zona sin asfaltar de manera dinámica.

***Pedido**:* Es la clase que permite al usuario elegir sus restricciones a la hora de buscar un vehículo apropiado para su viaje

***Viaje**:* La clase implementa IViaje y es utilizada para qué armar el viaje apropiado para el usuario, seleccionado un chofer,vehiculo, marcando el viaje como solicitado

***ViajeFactory**:* Clase utilizada para aplicar el patrón Factory a la clase Viaje

- **Desarrollo de ejecución (Seguimiento de Prueba):**

Se instancia Sistema(aplicando patrón Singleton, pues solo hay una instancia). Se instancian los vehículos a través de la clase VehículoFactory(aplicando patrón Factory; parámetros: String indicando el tipo de vehículo y String de patente). Se agregan al ArrayList del Sistema.

Se instancian clientes, que contienen: nombre de usuario, contraseña, Nombre, Apellido, Telefono, email, dirección(Clase que contiene nombre de calle, altura, piso, letra) y fecha de nacimiento. Luego se procede a abrir un try/catch.

Dentro del **try** se agregan los clientes dentro del ArrayList de clientes del Sistema. En el **catch** atrapa la excepción de “Usuario existente” si es que hay 2 nombres de usuario iguales. Muestra mensaje por pantalla indicando la excepción mencionada. Se sale del try/catch.

Se instancian los choferes, hay dos de cada tipo de chofer. Se agregan los choferes al ArrayList de choferes del Sistema.

Se instancian los pedidos; parámetros del constructor de pedido: Fecha de pedido, zona, mascota, cantidad de pasajeros, equipaje, Cliente que solicita el pedido, distancia a recorrer(se asume que es en **kilómetros**. Se procede a abrir un try/catch,

Dentro del **try**, se instancian los viajes. Se utiliza el método **asignarPedidoVehiculo**, instanciando el viaje con el patrón Factory. Se valida el pedido con el método **validarPedido**: se revisa si la cantidad de pasajeros es menor o igual a 10 (si supera, lanza excepción), también se revisa que si se supera los 4 pasajeros y se pide mascota, lanze excepción(porque tendría que ser una combi, y las combis no permiten mascotas). Se consulta si hay vehículo disponible (puede suceder que no haya vehículos porque se están usando todos, se lanza una excepción). Se busca el mejor vehículo a través del método **buscarMejorVehiculo** (se llama a la función **getPrioridad**, donde se evalúa cual es el mejor vehículo para el pedido solicitado, puede retornar null en caso de que no haya vehículos disponibles, en este caso lanza una excepción). El viaje queda en estado “Solicitado”.

Se le asigna el viaje al chofer con el método **asignarViajeChofer**. Toma del ArrayList de choferes al 1er chofer disponible. En caso de no encontrar choferes disponibles, se lanza una excepción.

El último paso es pagar y finalizar el viaje. El método **pagarViaje** es llamado por el cliente del viaje. Llama al Sistema(ya que es una sola instancia, por patrón Singleton) y usa el método **pagarViaje** del Sistema(hace que el viaje llame al método **pagarse,** donde setea el estado de viaje en **pagado**).Por último califica al chofer con el método **calificar_Chofer**(le da un valor del 0 al 10 de manera aleatoria). Por último, se toma el chofer del viaje y llama al método **finalizarViaje,** donde cambia su estado de ocupado de “true” a “false”, aumenta su cantidad de viajes y llama al método **finalizarViaje** de Sistema, donde se cambia el estado del viaje a **finalizado**, se saca de la lista al chofer y se lo agrega al final, lo mismo para el vehículo(también cambia su estado a **desocupado**).

- **<u>Muestra de listados:</u>**

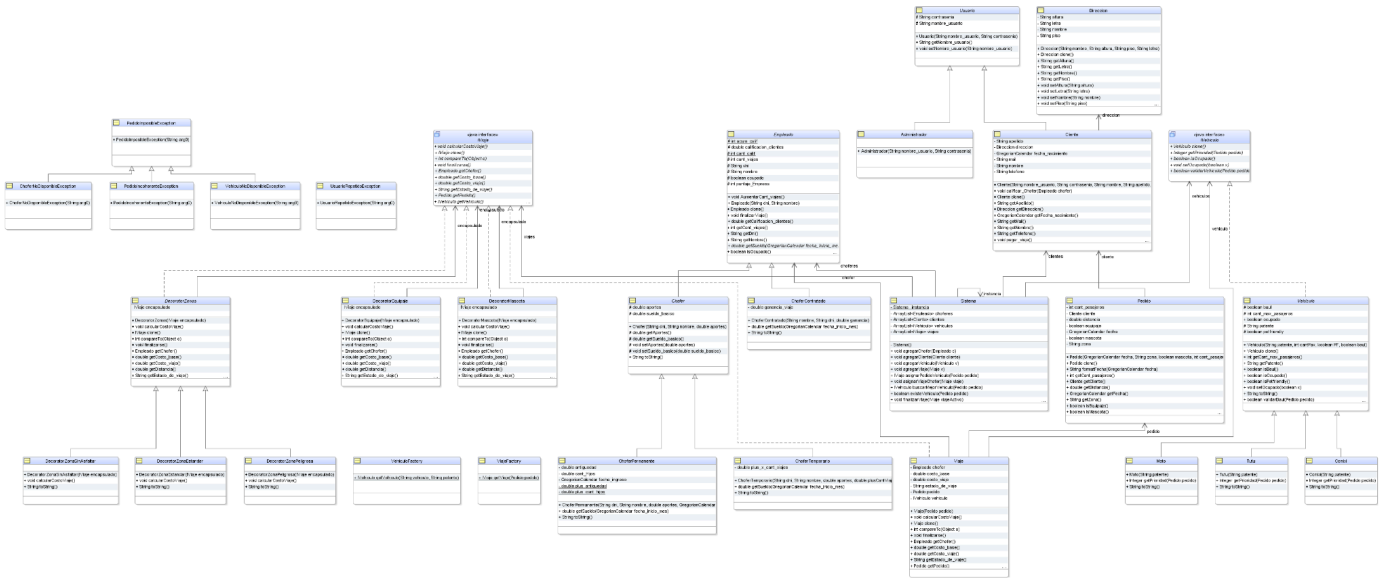
Dentro de la clase Prueba luego de la ejecución del programa se encuentran los listados de Choferes, Vehículos, Clientes y Viajes. También se hace un ejemplo de las funcionalidades del administrador:

- Saber cuanto es el monto a pagar por todos los sueldos del mes de todos los choferes. Como parámetro se ingresa el primer día del mes a analizar.
- Teniendo un chofer y una fecha de inicio y otra de fin, mostrar todos los viajes en ese lapso. Como parámetros se ingresa el chofer, fecha de inicio y fecha de fin.
- Teniendo un cliente y una fecha de inicio y otra de fin, mostrar todos los viajes pedidos en ese lapso. Como parámetros se ingresa el cliente, fecha de inicio y fecha de fin.
- Mostrar un listado con los sueldos de cada chofer por mes y el precio total a pagar. Como parámetro se ingresa el primer día de mes a analizar.
- Asignar puntaje a choferes en ese mes. Como parámetro se ingresa el 1er día del mes a analizar.

A todas las funciones que solo tienen una fecha que corresponde al primer día del mes a analizar se debe a que los viajes están ordenados por fecha, por lo que la condición para recorrer el ArrayList es que la fecha del viaje sea mayor a la fecha inicial.

- **<u>Diagrama UML</u>:**

Se encuentra en: Grupo_01\MavenApp\TpFinalPrograma3_2024\model\models



- **<u>JavaDoc</u>:**

Se encuentra en la carpeta del proyecto(**GitHub**).